Modelling Self-managing Multi Agent Systems Using Norms

Elakehal, Emad Eldeen

Award date:
2015

Awarding institution:
University of Bath

Modelling Self-managing Multi Agent Systems Using Norms

submitted by

Emad Eldeen Elsayed Ahmed Elakehal

for the degree of Doctor of Philosophy

of the

University of Bath

Department of Computer Science

April 2015

ABSTRACT

Businesses have an increasing need for adaptive systems to face the challenges of today's complex and dynamic world. The Multi Agent Systems (MAS) paradigm offers principles for the building of complex distributed intelligent systems, hence, MASs are potentially strong candidates to aid the take-off of a new generation of smart business processes. The use of one or multiple agents to manage business processes offers access to MAS properties such as the ability to cooperate and coordinate and to manipulate and change the system behaviour autonomously, in both a reactive and proactive manner.

Modelling and developing agent based applications is not an easy task though, and successful development of industrial-strength applications requires the availability of comprehensive robust software engineering methodologies. Although, a number of MAS development methodologies have been proposed and are now available, none of them has reached the status of being the standard method for developing MAS, and majority of them are not accessible for business users with limited or no knowledge of MAS.

Following the Design Science Research Methodology, we develop a new MAS development methodology called Modelling Self-managing MultiAgent Systems (MSMAS). We propose MSMAS as a means to enable business users, as well as MAS specialists, to translate their ideas into designs with embedded system norms. Norms are a type of constraints that forbids or requires certain types of behaviour within a context. The use of norms and their formal representation enable the software designers to verify the correctness and other properties of these designs, to map their designs into code ready for deployment, and to validate the behaviour of the running system against the requirements.

Our proposed methodology combines business process concepts with agent concepts and define notations for visual models with underlying formal presentation to build MAS applications. MSMAS thus attempts to bring multiagent techniques closer to real-world applications and commercial users.

This thesis covers our main contribution of defining a new methodology for the development of multiagent systems. The new methodology is established based on three fundamental aspects: concepts, models, and process. We present a set of agent and business process concepts in the form of metamodel, that supports the formalisation of MAS models, adopts the principles of normative multi agent systems, and employs the concepts of institutions and in-

stitutional roles to specify the organisational structure of such systems. We define four types of system norms to express the requirements at the level of system goals, institutional roles, communication protocols and business activities. MSMAS system norms are constraints that specify permissable and forbidden actions.

We realize our methodology by extending the semantics of an existing declarative language to express the system norms within MSMAS models. We also utilise existing logic-based languages to formally encode the system norms. We use then this representation of norms to verify the correctness and other properties of MSMAS system models. Furthermore, we present a mechanism for the monitoring of system traces to verify that the execution meets the requirements.

In conclusion, MSMAS is a comprehensive MAS development methodology that covers the full development life cycle and consists of three phases: analysis, design and implementation. MSMAS employs system norms and an institution structure to define the social aspects of the system and to capture the business requirements formally. MSMAS allows for verification and validation of the design and deployed system respectively. The evaluation of MSMAS against software engineering principles and the investigation of its strengths and shortcomings by means of comparison with a number of other MAS methodologies establishes, its capacity to capture business requirements and present them in visual models, with an underlying formal presentation that links business process concepts with agent concepts at a range of abstraction levels.

ACKNOWLEDGEMENTS

In the Name of Allah, the Most Gracious, the Most Merciful ”Rather, worship [only] Allah and be among the grateful.” : Quran - Surat Az-Zumar 39:66

All the praises and thanks be to Allah Almighty, the Giver of bountiful blessings and gifts. And prayers and peace of Allah be upon the noble Prophet Mohamed.

I wish to express my deepest gratitude to my supervisor Dr Julian Padget of The University of Bath - United Kingdom for his continuous guidance during the course of my research project and the preparation of this work. He has helped me in shaping the research from start, and have offered enormous amount of encouragement and support especially through the inevitable research setbacks. Without his support, this dissertation would not have been possible. I wish to say a heartfelt thank you to him.

Special thanks the Engineering and Physical Sciences Research Council (EPSRC) and The Book Depository Ltd for providing funding during the course of my research. In addition I would also like to thank Andrew Crawford, and Stuart Felton of The Book Depository limited for their support and encouragement since day one. Huge thank you to Kieron Smith of The Book Depository Ltd for helping in proofreading parts of this thesis and for his care and support. Likewise, thank you to Haitham Fatehy of Elkotob.com and, Ayman Abdul-Rahman and Khalid Diaa of Vijua.com

My sincere thanks to Dr. Marco Montali of Free University of Bozen-Bolzano - Italy, for his contribution to some publications related to this research and his invaluable feedback on some parts on this thesis. I would like also to express my gratitude to all research students and staff at the department of Computer Science at the University of Bath who have provided support. Special thanks to Dr Marina De Vos, Dr Owen Cliffe, Mesar Hameed, Dr Shadi Basurra, and Dimitris Traskas for their encouragement and fruitful discussions.

I would like to thank my family in Egypt for all the love and support they have provided throughout my life. My mother Firdaus Elshiemy, my sisters Thoraya and Tahany, and my brother Alla Eldeen are always incredibly supportive and I am very proud to have all of you.

Last, but certainly not least, my smaller family who have always been, and continues to be there for me. My wife Agnija Elakehal have been amazingly supportive. And my little sons, Ahmed and Gabriel-Mahmoud, thanks for your love and patience. (Emad Eldeen Elakehal, Feb 2014)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION AND MOTIVATION

Today's world has grown more complex and more dynamic, largely as a result of advances in technology and communications that increase connectivity and an expectation of shorter response times. Businesses face a range of new challenges in consequence: they must adapt to change rapidly, need to change goals, operations and processes very rapidly and at low cost. Business agility can be preserved to some extent by maintaining and adapting goods and services to meet customer demands, and adjusting these to the changing market. But sometimes, change is so fast and so frequent that traditional organisations and traditional software systems are unable to meet the challenge.

A need for adaptive systems is evident when we observe the new trend of so-called smart business processes, a new wave of business process which is more proactive and ready to change its behaviour – either the better to interact with other business processes, or to manage its resources. Advances in Business Process Management have included designing flexible, easy to change business processes (BPs), but managing these processes has still been done in a traditional manner, with means of service discovery and enactment. These new processes need more than just the ability to change, they need to be manageable – either by themselves or by more capable entities that oversee them.

The questions then are: how can we build software systems that are able to change their behaviour? How can a system automatically perceive the events in its environment, and filter these so as to look at only those events that are significant to it and relevant to its business goals? How can a system reason about the changes and reach the conclusion that its behaviour and processes are no longer suitable? How can it set for itself new business goals and plan new sets of processes that respond to the changes, and still fulfill these new business goals? How can the system translate these plans to actions? How can it acquire new capabilities to execute the planned actions if it is not currently capable of achieving its plans? Agile software development methodologies [Highsmith and Cockburn, 2001] appeared as a response and solution to this modern requirement, and a large number of enterprise organisations recognised the

need to adopt and adapt these methods. But traditional software modelling paradigms remains somehow static, design principles stay the same, and the overall mindset of developing software systems using these paradigms remains largely unchanged. The limitations of traditional software development methods, being centered around the object model or interacting system components rather than rational decision making and social system components [Wooldridge, 1997], suggests a need for a fresh paradigm. Multi Agent Systems (MAS) were there even before the demand for ever-changing software systems had emerged. This relatively modern approach for modelling and building complex systems started attracting some attention in the past decade but was still not widely used.The use cases for these systems were, not surprisingly, exploratory and somewhat limited in the early years and the examples were not convincing as a new wave of programming paradigms.

MAS' main aim is to provide principles for the building of complex distributed systems that involve many agents that are able to interact autonomously using a range of mechanisms for cooperation and coordination. Agents' action sequences would then not need to be programmed explicitly beforehand; instead, agent behaviours can be coded in the form of goals, and constraints that define their permissable and forbidden actions, to enable agents to figure out on their own how to achieve these goals in the context of a changing environment. Effectively, an agent needs to work out what to do by assessing its environment and reasoning about it. Large problems can be solved then by using a group of agents as a collaborative problem-solving system. New requirements can then be perceived by the agents as changes in their environment, leading to change in system behaviour and processes.

However, building multi-agent applications for complex and distributed systems is not an easy task [Edmunson et al., 1992]. Indeed, the development of industrial-strength applications requires the availability of appropriate software engineering methodologies. A number of MAS development methodologies have been proposed and are now available, however we believe none has reached the status of being the standard method for developing MAS, especially at a commercial level. MAS currently have all the problems of traditional distributed, concurrent systems, plus the additional difficulties that arise from flexibility requirements and sophisticated interactions [Wood and Deloach, 2000]. As a result, there is a real difficulty in defining an effective MAS development methodology. According to Luck et al. [2003]:

> "One of the most fundamental obstacles to the take-up of agent technology is the lack of mature software development methodologies for agent-based systems. Clearly, basic principles of software and knowledge engineering need to be applied to the development and deployment of multi-agent systems, but they also need to be augmented to suit the differing demands of this new paradigm".

Thus, there is a real need to define a MAS development methodology that not only satisfies theoretical requirements, but is also sufficiently practical to allow broader uptake of agent technology.

The lack of practical MAS development methodologies means this paradigm remains largely unexploited, and the demand for adaptive systems is therefore not yet fully met. The use of this paradigm in commercial applications is very limited [Müller and Fischer, 2014] and often poorly implemented – only progressing thanks to efforts of enthusiastic users to find tools and processes which are ready to use, to translate their ideas into designs, and to map their designs into code ready for deployment. The task of our research project is to define a new methodology that is able to overcome the disadvantages of current methodologies, and be equally accessible to system modellers and designers of business process management systems, as well as experts in multi-agent systems, who may use it to specify and develop simulations and prototypes more quickly, through working at a higher level. This methodology should enable these users to design new systems quickly, specify their requirements using visual models, verify the correctness of their designs, and present their system specifications in a formal representation for implementation.

Our work aims to help a wide range of interested parties in understanding the benefits of the agent paradigm, and in learning how to use them to structure their applications without the need to be full experts in MAS, or have to face the challenge of writing complex mathematical or logical formulae.

Our work is an attempt to bring multiagent techniques closer to real-world applications and commercial users, in response to one of the identified major shortcomings of the discipline as stated in 2003 in the Agentlink Roadmap, specifically that "One of the most fundamental obstacles to the take-up of agent technology is the lack of mature software development methodologies for agent-based systems." [Luck et al., 2003]. We believe this still remains largely true today.

1.1 Problem Statement

A close look at existing MAS development methodologies shows that none of them has become the dominant method to use, for a wide range of reasons. Here we list those reasons we regard as the most crucial:

1. *Support for Inexperienced Developers:* The majority of current MAS development methodologies require deep knowledge of agent concepts, some of which are abstract, poorly understood, or lack common agreement on how to implement them. Developers need to specify all the components of their agents and think of how their social aspects could be designed and implemented. Consequently, commercial applications are rarely developed using the MAS paradigm. And although some efforts were directed to develop methodologies that support engineers with limited or no experience of agents such as the work of [Bussmann, 2003], who proposed an agent-oriented methodology to design production control systems, that methodology is limited only to that application domain and requires deep knowledge of production control. Hence it is no suitable for general

users or any other application domain.

2. There is a lack of agreement between different MAS experts about what MAS really constitutes [Bordini et al., 2007], which concepts it covers and what each concept means [Dastani et al., 2004]. This is due to the field being under development and because there is still a variety of terms for the same concept, as well as different concepts using the same terms. As a result, there is no one single metamodel that is used across research and development activities, making it impossible to transform modelled systems between the various platforms.
3. *Absence of an holistic view:* many of the current methodologies focus on supporting one or a small number of related aspect(s) of modelling the system Sturm et al. [2003]: some focus on the organisational, others on the system logic and cognitive aspects, and others on the definition and specification of the requirements or the formal presentation of the system specifications. This diversity has potentially contributed to confusion and ambiguity about the purpose of MAS and what kind of system can be designed and developed as a MAS.
4. *Lack of Whole Life-cycle Coverage:* Some of the current methodologies provide a designer tool to facilitate its process.Most of these tools support only the design and analysis phase, and none of them supports the deployment and system verification [Sabas et al., 2002], others offer theory without supporting tools, while others have full coverage of the life-cycle but no guidelines or lack of a commercial-strength integrated development environment.
5. *Gap Between Design and Implementation:* There is an obvious gap between the design models and the existing implementation languages [Sudeikat et al., 2004], which leads to substantial difficulties in trying to map complex designs into executable code.
6. *Absence/Presence of an Implementation Phase:* Most of the current methodologies do not include an implementation phase. One that does is TROPOS [Bresciani et al., 2004], but it does not explain how to implement beliefs, goals and plans, nor reasoning about agent communications [Dastani, 2004].
7. *Limited Formal Representation of MAS Concepts:* Despite early work by Wooldridge [1992], and Luck et al. [1996] neither of these has resulted in full formal models, and the recent work of d'Inverno et al. [2012] although it offers a full formal model, it covers only those concepts related to electronic institutions. Even though a partial approach may be effective, the question remains, which concepts to formalise? And what is the best way to specify and describe a MAS?
8. Although the concept of software development methodology is clearly defined in the field of software engineering, there is no common structure and components that are used across the current MAS methodologies, they differ between themselves in terms of phases, steps, and tools.
9. Most of the software engineering aspects such as accessibility, preciseness, expressive-

ness, traceability, refinement, definitions, and modularity are not supported comprehensively across any current methodology, which leads to problems of industry acceptability and a consequent negative image of agent technology [Dam and Winikoff, 2004].

10. Some of the current methodologies are defined based on Object Oriented Programming concepts, which consider agents as just complex objects, hence they lack support for MAS organisational structure and agent communications aspects [Bush et al., 2001].
11. Most of current methodologies lack the support for verification and validation of the designed and deployed systems. Moreover the methodologies that support these functions use verification only on code or use traditional debugging methods such as the work of Padgham et al. [2005]. Others use formal model checking *only* during design time. While debugging is useful technique, it cannot guarantee the correctness of a system, so formal verification techniques become extremely important [Bordini et al., 2007].

These reasons collectively, lead to the definition of our research problem, which is that current MAS development methodologies are not able to support the increasing demand for building agent based systems at a commercial level. And that both businesses and academic research in the field of MAS are missing opportunities to develop because the majority of current complex use cases do not get to be modelled using MAS concepts, which means the research of MAS is not challenged with real world cases that helps it to develop new methods and techniques.

1.2 Research Objectives

The main aim of this research is to improve multiagent systems development practice and bridge the gap between academic and industrial applications by offering a new methodology for developing multiagent systems, whose objectives are to:

1. Define a set of requirements for MAS development methodology that overcome the drawbacks of current methodologies and bridge the gap between theory and practice.
2. Select a suitable organisational structure for MAS that is suitable for most use cases and identify the necessary concepts to specify that structure.
3. Define a set of agent concepts in the form of a metamodel, that supports the formalisation of MAS models and allow for transformation from one modelling method to another.
4. Define a detailed development methodology that uses the defined MAS metamodel and employs a number of steps to allow for design, verification and implementation of MAS.
5. Select a formal language to describe the designed system, and show how to verify the system models at design time as well as how to utilise the formal specifications to monitor the execution of an implemented system to obtain a degree of self-management.
6. Evaluate our methodology through comparison with other methodologies to assess its strengths and to establish that it succeeds in meeting the objectives expressed here.

1.3 Thesis Contribution

This thesis makes five main contributions:

1. A metamodel that combines a group of concepts linking business processes with system goals and specifies the organisational structure through the use of norms. The metamodel is expressed in UML[1], where it displays our selection of concepts and their relations. We complement the UML diagrams with detailed descriptions of each concept and compare our metamodel to other related metamodels to highlight, for each group of concepts, the corresponding parts in other metamodels.
2. The principle of a set of system norms, that constrain the system behaviour, at the level of system goals, institutional roles, communication messages and business activities. We extend the semantics of visual notation of ConDec^{++} [Montali, 2010], a declarative formal language based on DECLARE/ConDec[Montali, 2010, Pesic et al., 2007, Pesic and van der Aalst, 2006a], and we map our defined system norms to a formal logical presentation in CLIMB [Montali, 2010].
3. A new development methodology that overcomes most or all of the issues identified with current MAS development methodologies.
4. The mapping of MSMAS norms to existing formalisations, namely CLIMB and $EventCalculus$ [Kowalski and Sergot, 1989] to enable the verification of MSMAS models at design time using CLIMB framework [Montali, 2010], and $\mathcal{S}$CIFF and g-$\mathcal{SCIFF}$ proof procedure [Alberti et al., 2008], and at run time using the reactive $EventCalculus$ framework.
5. A set of guidelines for designing MAS using MSMAS, verifying designed models, implementing the system and monitoring and verifying the execution traces during run time.

The proposed methodology (MSMAS) is evaluated against software engineering principles and compared against other methodologies to evaluate its strengths and shortcomings. The research achievements are discussed and summarised and a set of emerging research questions are identified for future work.

1.4 Research Methodology

Given the presence of qualitative aspects in the research problem, we felt it was desirable to select a research methodology that is able to address the approach and guidelines and that can lead to answering the research question(s) and validate and evaluate the research outcomes and verify our hypotheses. In our research we have followed the method proposed by Peffers et al. [2007], called the Design Science Research Methodology (DSRM), that is well grounded in existing literature about Design Science in Information Systems. In their methodology they provide guidance and a mental model for the presentation of the research project and its out-

[1] Unified Modelling Language

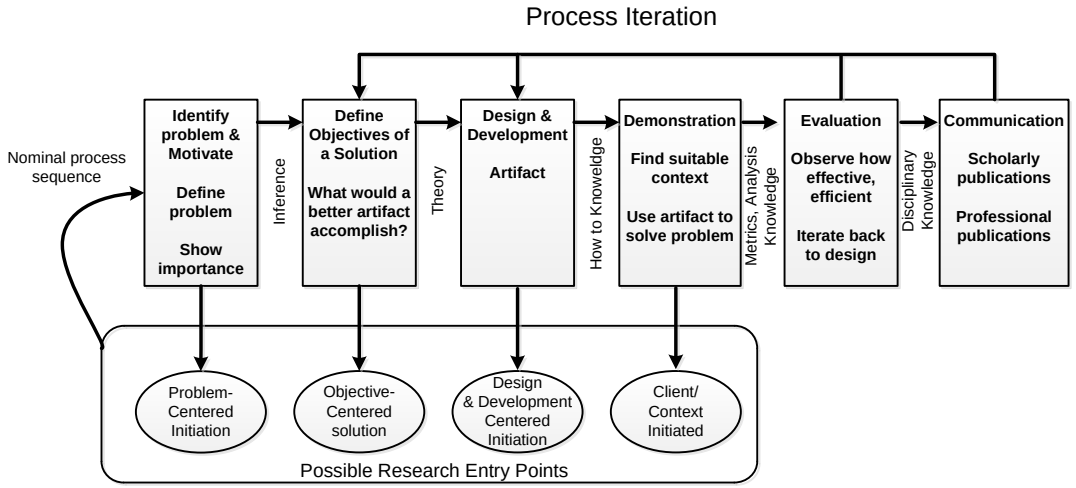


Figure 1-1: *DSRM Process Model (used with permission) [Peffers et al., 2007]*

comes.

The DSRM process consists of six different activities, as shown in Figure 1-1, and listed briefly below:

Activity 1: Problem identification and motivation: This activity addresses defining the problem that the research aims to solve and the value of the solution. The problem definition will influence the development of artefact that can effectively provide a solution. Justifying the value of a solution is important as it serves as a motivation for the researcher and the audience and it helps to understand the reasoning associated with the researcher's proposed solution. The identified research problem and questions can be then explicitly transformed into system objectives or what is called meta-requirements.

Activity 2: Define the objectives for a solution: This activity concerns reasoning about the identified problem and stating explicitly the objectives of a solution based on a view of what is possible and feasible. The expression of objectives can be quantitative, e.g. describe how a desirable solution would be better than current ones, or qualitative, e.g. describe how a new artefact is expected to support solutions to outstanding problems.

Activity 3: Design and development: This activity focuses on the creation of constructs, models, methods, instantiations or new properties of technical, social, and/or informational resources. Following that, an evaluation of the solution is carried out either by means of demonstration or through a formal evaluation of the developed artefact.

Activity 4: Demonstration: This activity demonstrates the use of the artefact to solve one part or all of the identified research problem. Demonstration may involve an experiment, simulation, case study, proof, or other appropriate activity.

Activity 5: Evaluation: To assess how well the artefact supports a solution to the research problem through observation and measurement. This can be done by comparing the objectives of a solution to actual observed results from use of the artefact in the demonstration.

Activity 6: Communication: The final activity concerns communicating and disclosing the resulting knowledge. This is done by communicating the problem, its solution, its utility and novelty, and the effectiveness of this solution to other researchers as well as wider audiences.

Although, this process is structured and stated in a sequential order; the reality of a research project is that starting with activity 1 is always true then proceeds following the process sequence in many iterations, each one covering one aspect of the solution and, through the journey articulating the solution, it is common that earlier parts are revisited and adjusted to take account of new knowledge.

Following this process as described above, and after the definition of the research problem and questions, our plan comprises:

1. Survey the related work in areas of software engineering, business processes and normative multiagent systems. This activity includes understanding the following concepts (component based software design, formal language specifications, logical programming, model based software engineering, ... etc).
2. Study the existing agent software development methodologies and the supporting software development tools, if any. The study focuses on discovering the strengths and weaknesses of each methodology. We have selected a group consisting of well-known methodologies depending on recognised evaluation results to check that our understanding of each methodology matches other researcher's understanding in terms of where and how it stands in comparison to other methodologies.
3. Specify a set of requirements for the proposed software development methodology at the process, concepts, and models levels.
4. Design and develop the methodology artefacts and processes that overcome the issues identified in other methodologies, and support the identified requirements. We use a case study as a guiding problem to assess how our chosen concepts and designed models satisfy the requirements.
5. Demonstrate the suitability of our design by applying it to a real world case study. We model a system using the graphical notations we have identified as well as expressing the specifications formally.
6. Assess and evaluate the artefact through gaining feedback from experts in the field through submitting reports on our work to academic venues that offer peer-review process.

We have gone through these steps many times until we have built a complete methodology and subjected it to evaluation using a feature analysis and evaluation framework, which is a common practice for this kind of research project in software engineering.

1.5 Thesis Structure

The thesis consists of eight chapters:

Chapter 1 covers the motivation and the statement of the problem, as well as the main objectives of the thesis and the contribution. This chapter sets out and explains the rationale behind the development of a new multiagent system development methodology. It lists a number of issues within the existing MAS methodologies and it defines the research objectives and the contributions of our work.

Chapter 2 explores current research including the concepts of Multi Agent Systems, review of a selection of current MAS development methodologies as well as a section of MAS metamodels, business processes management and modelling, norms, institutions and review of related worn on normative systems, and finally a review of self-management in Multiagent systems.

Chapter 3 starts with our list of requirements for building a multiagent development methodology, followed by a presentation of the main concepts that we regard as the minimum set of concepts needed for designing and specifying any MAS and the rationale behind their selection. Then a detailed a presentation of these concepts and their sub-concepts in the form of a new metamodel. All segments of the metamodel are presented in detail and compared to concepts found in the related work reviewed earlier in Chapter 2.

Chapter 4 starts with the presentation of our proposed methodology, MSMAS, detailing its three phases, (System Requirements – System Design – Implementation). It present the methodology visual models and their notation accompanied with a set of simple examples. We present wherever applicable each group of our defined system norms with their graphical notation and their semantics. The chapter ends with a set of guidelines for using MSMAS.

Chapter 5 is a detailed case study that illustrates by example how MSMAS works in practice. The case study demonstrates MSMAS features and models and show our recommended approach for modelling self-managing systems.

Chapter 6 starts with a brief discussion of the importance of using formal models and which formalisms to use, followed by a summary of our chosen formal language CLIMB for the support of static verification of MSMAS norms. The translation of MSMAS norms into CLIMB is detailed, followed by an example of how MSMAS models can be verified at design time. The second part summarises our chosen framework for online monitoring: the Event Calculus – we present the translation of MSMAS norms into Event Calculus and show by example how to monitor a MSMAS model during execution.

Chapter 7 presents an evaluation framework and evaluates and assesses the MSMAS method in comparison to selected methodologies. The discussion part of this chapter explores the advantages of MSMAS and its weaknesses in relation to other methodologies.

Chapter 8 outlines the conclusions of our work. The chapter revisits the objectives and discusses our approach and how we achieved them. We conclude with recommendations

for future developments and possible topics for research in relation to our findings.

1.6 Related Publications

The following list includes all papers published by the author which are related to this dissertation. In each case my contribution to the paper is stated in accordance with regulation 16.1 subsection 3.v of the University of Bath regulations.

1. [Padget et al., 2014] On Requirements Representation and Reasoning Using Answer Set Programming - Julian Padget and Emad Eldeen Elakehal and Ken Satoh and Fuyuki Ishikawa. Submitted to the first International Workshop on Artificial Intelligence for Requirements Engineering (AIRE 2014) and Published by IEEE.
   This paper presents an approach to the representation of requirements using Answer Set Programming (ASP). The represented approach includes: (i) a metamodel that incorporates the notion of runtime requirements,(ii) a formal language for their representation and its supporting computational model (InstAL), and (iii) a software component that enables monitoring in distributed systems. My contribution to this paper is limited to the report on the commercial applications we developed during the course of my research, and a brief representation of MSMAS metamodel that utilises the concept of norms to represent the regulations imposed on the system constructs. Both contributions are discussed in detail in Chapter 4 and Chapter 5.
2. [Elakehal et al., 2014] Run-time Verification of MSMAS Norms Using Event Calculus - Emad Eldeen Elakehal and Marco Montali, and Julian Padget. Submitted to The international workshop on Quality Assurance for Self-adaptive, Self-organising Systems (QA4SASO 2014). This paper focuses on the problem of online verification and presents our approach to allow MSMAS system designers to verify their system during execution. In this paper we have presented our defined norm types - that MSMAS uses - these are detailed in this thesis in Chapter 4. Other contributions of this paper include the modelling of MSMAS norms life cycle as non-atomic activities and their formal representation in Event Calculus. Finally we illustrated the suitability of our approach for the purpose of run-time verification through an example. An extended version of the example can be found in this thesis in Chapter 6 Section 6.6.

3. [Elakehal et al., 2013] Verifying MSMAS Model Using $\mathcal{S}$CIFF - Emad Eldeen Elakehal and Marco Montali, and Julian Padget. Published in Lecture Notes in Computer Science Volume 8076, 2013, pp 44-58, June 2012. Presented in MATES 2013 in Germany

   This paper addresses the problem of static verification of the designed models and presents our technique to allow MSMAS system designers to verify their models during design

time. In this paper we have presented our defined norm types - that MSMAS uses - these are detailed in this thesis in Chapter 6. Other contributions of this paper include the representation of the system norms in SCIFF and the explanation of how to use SCIFF reasoning capabilities to verify formal properties of MSMAS models. More details on static verification and the formal representation can be found in Chapter 6 Section 6.3.

4. [Elakehal and Padget, 2012b] MSMAS: Modelling Self-managing Multi Agent Systems - Emad Eldeen Elakehal and Julian Padget. Published in SCPE: Scalable Computing: Practice and Experience, Volume 13, Number 2, June 2012.

   This is an early presentation of MSMAS methodology, where our contribution includes the presentation of MSMAS concepts and its three phases and the presentation of an early version of MSMAS metamodel, that was done before considering capturing the system requirements formally.

5. [Elakehal and Padget, 2012a] Market Intelligence and Price Adaptation - Emad Eldeen Elakehal and Julian Padget. Presented at the 14th International Conference on Electronic Commerce 2012 - Singapore, Singapore.

   In this paper we report on a commercial Multi Agent System that was designed and developed to support the business of an online book retailer. The case study presented in Chapter 5 is based on this system.

6. [Elakehal and Padget, 2011] A Practical Method for Developing Multi Agent Systems: APMDMAS - Emad Eldeen Elakehal and Julian Padget. Invited paper in IDC 2011 : 5th International Symposium on Intelligent Distributed Computing - Delft, Netherlands. This is an early report on MSMAS methodology (called APMDMAS at that time), the contribution of this paper included the identification of a number of issues in the current MAS development methodologies that motivated us to develop a new methodology. The paper has an overview of the methodology's three phases as well as a number of example visual models. The details of the methodology can be found in this thesis in Chapter 5.

7. A Metamodel for Self-managing Multi Agent Systems - Emad Eldeen Elakehal and Julian Padget. Presented at 13th International Workshop on Agent-Oriented Software Engineering - Valencia, Spain. This is an early version of a metamodel to support MSMAS, in this paper we have presented a number of concepts and how they relate to each other. The final version of our metamodel can be found with details in Chapter 3.

8. [Elakehal and Padget, 2008] Pan-supplier Stock Control in a Virtual Warehouse (Industry Track) - Emad Eldeen Elakehal and Julian Padget. Presented at the 7th International

Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008). Presented in Estoril - Portugal.
In this paper we report on a MAS application that we designed. The case study presented in chapter 5 is based on this application. A short version, was also published and presented at the 6th European Workshop on Multi-Agent Systems December 2008 - Bath, United Kingdom

1.7 Chapter Summary

The MAS paradigm is a strong candidate to respond to the increasing demand from enterprises for adaptive and flexible systems. Developing such systems requires the availability of methods and processes. One factor in the popularity of the MAS paradigm is how accessible these methods are. We have identified and listed a number of issues within the current MAS development methodologies that led us to aim in this work at developing a new comprehensive MAS development methodology that bridges the gap between academic and industrial applications. Our proposed methodology is called MSMAS and it uses a number of commonly-recognised agent and business process concepts, with underlying formal representations, and it employs the concept of system norms to capture requirements and to define the organisational structure of the system under development. MSMAS allows for verification during design time as well as monitoring the fulfillment of requirements during run time.

CHAPTER 2

BACKGROUND, CONCEPTS AND LITERATURE REVIEW

To gain the breadth and depth of knowledge necessary to address the issues identified in the current MAS development methodologies, we surveyed the literature of several disciplines, including software engineering, business process modelling and a range of topics related to MASs. This chapter is to present and discuss in detail each topic and review sets of specific related work in their respective sections. We start with a brief introduction, in Section 2.2 where we analyse the MAS paradigm including the definition of an agent and a multi agent system (MAS), a selected list of common MAS architectures, organisational structures and a classification of MAS methodologies. We also summarise five existing methodologies and discuss their strengths and weaknesses in Section 2.3. A detailed evaluation, by means of comparison, of four of these methodologies with MSMAS is presented in Chapter 7. Section 2.4 also include a review of a selection of MAS metamodels. Business process modelling (BPM) is the focus point of Section 2.5. We present business process management concepts including the concept of smart business processes and agent-oriented BPM. In Section 2.6 we present the concepts and definitions of Institutions and Norms in MAS and discussion and presentation of our findings from literature in Section 2.7. We end this chapter with the definition of self-managing systems and briefly present a number of approaches for enabling a degree of self-management in MASs.

2.1 Introduction

Concurrency is a term we use to refer to the sets of events that happen simultaneously. By nature, the real world is concurrent, where a large number of events happen simultaneously. Thinking of humans that carry out both simple and complex actions, such as walking or driving a car, we can expect large numbers are doing the same actions as well as different actions at the same time. In fact sequential activities are rare when it comes to real life, it is hard to imagine that there is only one action a time in any setting. When it comes to programming computer

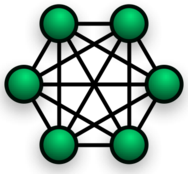


Figure 2-1: *A Fully Connected Network Example*

systems, and the advances of networking technologies and processing powers, concurrency is becoming the norm, where it is expected that multiple processes are executing at the same time. After a long time in which the vast majority of programming languages were sequential, where the underlying operating system provided for virtual concurrency, today both operating systems and programming languages are able to support concurrency. Modelling systems that are able to represent the real world or to interact with the real world, have become possible and Concurrency Oriented Programming Languages (COPL) have evolved to respond to this. COPLs, as proposed by Armstrong [Armstrong and Helen, 2003], however are built on the assumption that building concurrent systems can be done by:

1. Identifying all concurrent activities in the world we want to model
2. Identifying all message channels between the concurrent activities
3. Writing down all the messages which can flow on the different message channels.

Although this approach sounded sensible in its time, there are many issues that make it impractical. Some brief examples of these issues:

1. In a dynamic open world, it is hard to control who is participating and what are they doing, thus identifying all possible concurrent activities is not a feasible task. It will be hard to predict which activities at all times.
2. Assuming that we could come up with a list all possible activities, the task of defining all possible communications channels between these activities would become an even bigger task with the size of a fully connected network where the number of communication channels $C = \frac{n(n-1)}{2}$
3. Assuming the above two tasks are still possible, and the system is developed, this system remains static where all inputs and all outputs are known. Thus, if a change occurs in the environment, the system will require recoding.

MAS is seen as a better way to model concurrent systems and other classes of systems that require not only a representation of concurrent events but also a way to model their intra-relations and their effects. In the following section we present a summary of MAS paradigm and the classifications of MAS development methodologies.

2.2 Multi Agent Systems Paradigm

In this section we present our definition of agent and multiagent systems, the common architectures of MAS, and our classification of MAS development methodologies.

2.2.1 What is an Agent?

The central and most dominant concept in the MAS paradigm is the *Software Agent*. Many definitions have been given for what is actually a software agent, however the most commonly used definition is that given by Wooldridge and Jennings [1995a] who proposed two notions of agency: a weak notion and a strong one. The weak notion states that an agency is either hardware or software computer system that holds the following properties:

- **Autonomy**: where agents have a degree of control over their own actions and they can operate without any human intervention.
- **Social ability**: agents have the ability to communicate with other agents or humans through standard languages or protocols.
- **Reactivity**: where agents are able to sense changes in their environment and are able to respond to some of these changes.
- **Pro-activeness**: agents are also able to act not only to respond to external changes but also motivated by their own goals and internal states.

The strong notion uses mental components such as belief, desire, intention, and knowledge to define what an agent is and how it behaves. Considering the mental state of an agent indicates the autonomic nature of an agent in sensing and acting on a finite set of states of its environment [Wooldridge, 2008].

Following this definition and others, such as [Zambonelli et al., 2003b], [Henderson-Sellers and Giorgini, 2005] and [Odell, 2002], the key properties of software agent types amongst others are: an agent is *autonomous* which indicates that the agent possesses its own line of control on its actions and it may be able to move across different networks and platforms, the second property is *cooperative* which indicates that some of the agent's goals can exceed its capabilities and the agent may communicate and coordinate its behaviours with other agents to cooperatively perform the needed activities to achieve the goal, and the third property is its *learning* ability which enables the agent to act in a deliberative manner and compete with other agents based on its observations of its environment.

We define an agent as a software proactive system participant that actively assesses its internal state and internally plans and acts to achieve its goals. A full list of MSMAS concepts and their definitions is available in Chapter 3.

2.2.2 What is a Multiagent System?

A multiagent system is a system that is composed of several agents and in most cases is capable of achieving goals that are not possible to achieve by one individual agent acting alone.

Jennings and Sycara [1998] identified the following characteristics for a MAS:

- Each individual agent has incomplete set of information or capabilities to solve the main system problem
- There is no global control of the system
- Data is decentralised
- Computation is asynchronous.

Another view of MAS is if the system relies on or facilitates agents interactions regardless of how these interactions are used. Multiagent systems can show a degree of self-management and can be classified as either *Closed Multiagent Systems* or *Open Multiagent Systems*. In closed MAS there is a common communication language and each agent is developed to be an expert at solving particular problems and has particular skills, and knowledge. In open MAS there is no prior static design, instead independent agents operate without prior knowledge of agents or services. As a result open MAS should have a mechanism to identify agents, control and facilitate their interactions.

We view MAS as a system that contains a number of autonomous entities which are capable of achieving some or all common goals of the system through the use of planning, reasoning and/or communicating. Our definition of MAS in the context of this work appears in Chapter 3.

2.2.3 MAS Architectures

MAS are built on different approaches which specify how the system can be decomposed into a set of component modules and how agents are organised and interacting. There are five agent architectural styles which we list briefly as follows:

Reactive Architecture: This is the simplest architecture where there is no central world model and there is no use of complex reasoning [Wooldridge and Jennings, 1995b]. The primary goal of reactive agents is to be robust and responsive. One can argue that majority of agent architectures have one or more reactive components.

BDI Architecture: Belief, Desire and Intention is a representation of the information, motivational, and deliberative states of the agent under the assumption that the agent is a rational component that holds a certain mental attitudes of Belief (referring to the informative component of system state), Desire (the motivational state of the system), and Intention (the deliberative component of the system). The BDI was developed based on the work of Bratman [1987] and developed by Rao et al. [1995]. Many MAS modelling methodologies are inspired by and have taken BDI concepts as their core modelling concepts.

Planning Architecture: In this architecture agents depend on either predefined plans or dynamically generated plans to determine the actions they will perform. Plans also can be total order plans that consist of full list of steps leading to the achievement of a goal, or partial order plans that have some ordered steps while the remaining steps are inconse-

quential.

Knowledge Based Architecture: these also known as expert systems, where agents depend on data structures consisting of explicit representation of problem solving information. The knowledge in this architecture is a set of facts about the world. This type of systems excels at accepting new tasks if explicitly described, achieve their task results quickly, and can adapt to changes in the environment [Russell et al., 1995].

Deliberative Architecture: In this architecture there is an explicit model of the world, and agents make their decisions via logical reasoning. Building MAS following this architecture requires: a formal description the real world and building a mechanism that allows agents to reason about these processes based on the available information.

2.2.4 MAS Organisational Structures

Academics widely agree that a multiagent system is a set of autonomous agents that try in a proactive manner to achieve a set of individual and/or group goals. These agents, normally, decide freely on the best course of actions that leads to the achievement of their goals. Having self-interested agents that act freely without observing the global goals of the system might lead to a loss of global coherence [Isern et al., 2011], where the system components seems functional however the system as a whole is not able to achieve its global goals.

Dignum and Dignum [2012] give a simple definition of an organisation: "the organisation consists of a set of agents (together with their capabilities and abilities) and a set of objectives. In each moment, the state of the organisation is given by a certain state of affairs that hold in that state.".

Ferber et al. [2004] have examined different definitions of multi agent systems and the main features of organisations, they have defined the following:

1. An organisation is composed of a number of agents that display a behaviour.
2. An organisation may be seen as a number of overlapping partitions (groups).
3. Agents' behaviour is functionally related to the organisation.
4. Agents' activities could form patterns that can be captured in a taxonomy of roles, tasks or protocols
5. The types of behaviours are related through the relationships between roles, tasks and protocols.

Moreover Ferber et al. [2004] have defined three main principles to be observed when designing or specifying an organisational focused MAS:

1. **Principle 1:** The organisational level is about describing "what" to be done and not "how" to do it, in this sense the organisational level forms the structure of the organisation and uses norms or laws to specify the restrictions and/or permissions on the agents' behaviour.
2. **Principle 2:** The organisational level should not be concerned with the description of agents and their mental states. Instead it should just provide descriptions of expected

behaviours.

3. **Principle 3:** The organisation might be seen as number of units each of these organisational units allow its members to interact freely. Each of these groups could have its own boundaries where its members only know how to interact within the group but are not necessarily aware of other groups structures or its members means of interaction.

As a consequence of these principles they conclude that; an organisation is a dynamic framework for agents to enter a group and play a role. The organisation can support the true "open system" concept by leaving the agents' architecture unspecified (at the organisational level) at the same time building secure systems could be achieved through defining groups that function as "black boxes" that are not open to those agents that are not members of this group.

A detailed survey of Multi-Agent Organisational Paradigms appears in the work of Horling and Lesser [2005] who identified ten different types of organisations which we summarise in Appendix A.

One of the common misconceptions as pointed out by Jennings [1999] is that "agent-based systems require no real structure". While this is true in certain cases, most agent systems require to have a structure. Building the society is needed to reduce the system's complexity, to increase its efficiency, and to model the problem in an accurate manner. Without an organisation structure the interaction patterns are unpredictable and predicting the overall behaviour of system becomes very difficult, if not impossible, with no norms defining what patterns are acceptable and what are not. In general, agents that are designed by different designers cannot be trusted to interact without any problem, this is not possible without having some rules describing the primitives of communications and the architecture of agents and the organisation of the groups they belong to. Specifying such rules can be done using what is called system norms, which are discussed later in this chapter in Section 2.6, and employed within our methodology in Chapter 4.

2.3 Classification and Review of MAS Methodologies

One of the first attempts to classify agent software development methodologies is the work of Bauer and Müller [2004]. They conclude as per Figure 2-2, that all agent software development methodologies are based on one of the following three approaches:

1. Agent based methodologies: these focus on the social level abstraction of individual agents, agents' groups and organisations. Examples of methodologies that take this approach are Prometheus [Padgham and Winikoff, 2002], TROPOS [Bresciani et al., 2004], and Gaia [Wooldridge et al., 2000a].
2. Object Oriented Methodologies: following the success of Object Oriented Programming (OOP) techniques, many methodologies tried to expand OOP by including the notion of agency. Examples of these methodologies are MaSE [DeLoach et al., 2001], ODAC [Gervais, 2003], and DESIRE [Brazier et al., 1997].

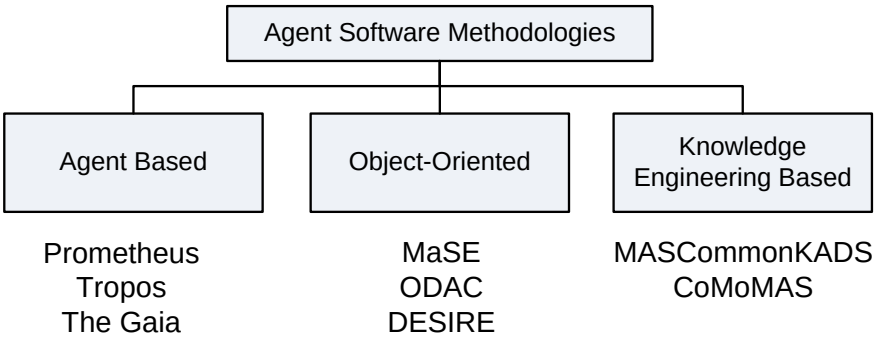


Figure 2-2: *Agent Software Methodologies Classification according to Bauer and Müller [2004]*

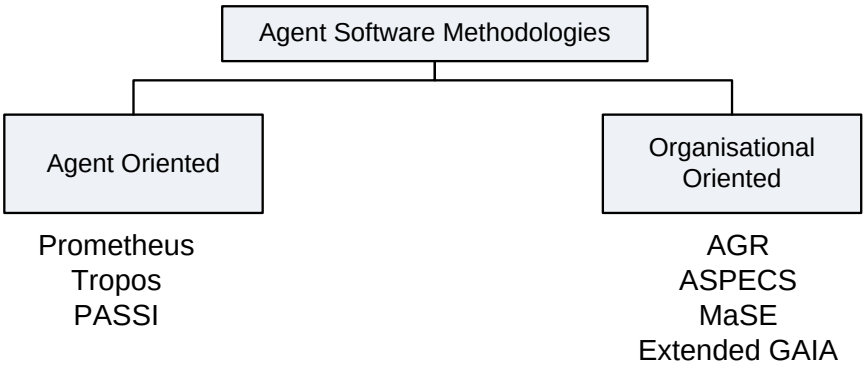


Figure 2-3: *Agent Software Methodologies Classification according to Argente et al. [2006]*

3. Knowledge Engineering based methodologies: these focus on modelling, identification and acquisition of the knowledge used by the agents. Examples of methodologies that use this approach are MASCommonKADS [Iglesias et al., 1998] and CoMoMAS [Glaser, 1997].

A more modern classification is the one proposed by Argente et al. [2006], who identified two families of methodologies as shown in Figure 2-3:

1. Agent Oriented: these focus on the design of each individual agent and its actions where MAS are designed around the agents' mental states following the BDI concepts (Beliefs - Intentions - Goals and Commitments. Examples of methodologies taking this approach are Prometheus [Padgham and Winikoff, 2002], TROPOS [Bresciani et al., 2004], and PASSI [Burrafato and Cossentino, 2002].
2. Organisational oriented: these are focused on the social aspect of the system structure where agents are normally seen as individuals belonging to one or more societies. They play a number of roles during their interactions with one another. This approach tries to model the MAS in same manner that reflects observed real life scenarios. Examples of methodologies that are based on this approach are AGR [Ferber et al., 2004], ASPECS [Cossentino et al., 2010], MaSE [DeLoach et al., 2001], and Extended GAIA [Wooldridge et al., 2000a]

In the following subsections we review a selection of the current MAS development methodologies.

2.3.1 GAIA Methodology

The GAIA is a general methodology that supports both micro (agent structure) and macro (organisational structure) development of agent systems. It was proposed by Wooldridge et al. [2000a] and subsequently extended by Zambonelli et al. [2003a] to support open multi-agent systems.

Figure 2-4 shows GAIA's key stages and its various models; the extended version of GAIA developed based on various organisational abstractions which need to be specified in the analysis phase; that allows for the system design process to start. The analysis process includes the identification of:

The organisation's goals which make up the overall system. The goals are a representation of the decomposition of the global organisation into loosely coupled sub-organisations.

The environmental model is an abstract description of the environment where the MAS will reside.

The preliminary roles model identifies the required basic skills within the organisation. At this stage, the notion of roles is abstract from any mapping into agents and should be limited only the the generic roles specification that is independent from any specific organisational structure.

The preliminary interaction model to identify the basic required interactions associated with the preliminary roles at an abstract level and independent from the organisational structure.

Organisation rules are those rules that need to be enforced across the organisation. The specification of these rules is important to ensure efficient execution and are normally expressed as constraints on the execution of activities or on the organisational roles.

The output of the analysis phase leads to the next design phases, which are the architectural design phase and the detailed design phase.

The architectural design phase includes:

The definition of the system's organisational structure by defining its topology and the control style. This is normally done by choosing an organisational structure from the number possible structures available in the catalogues of organisational patterns. The definition of organisational structure should consider: the organisational efficiency, the real-world organisation where the MAS is to operate, and the need to enforce specific organisational rules.

The completion of the role model based on the adopted organisational structure of previous steps. This is a kind of fine tuning of the roles and it includes as well the separation of organisational-independent aspects from the the organisational-dependent ones.

The completion of the interaction models would follow the previous step and be done in

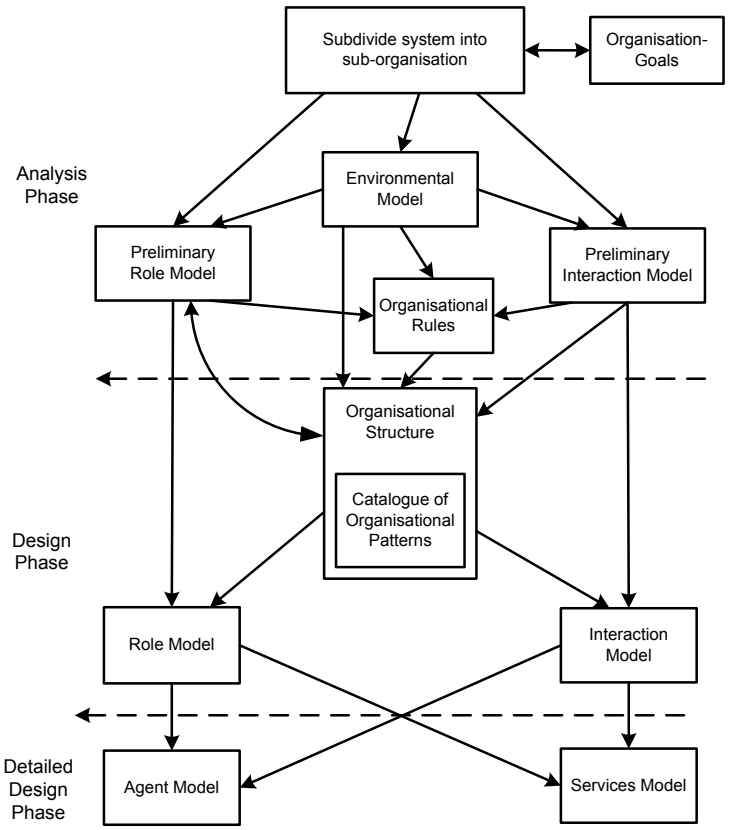


Figure 2-4: *Overview of GAIA Methodology Models (used with permission) [Zambonelli et al., 2003a]*

light of the selected organisational structure.

Next is the the detailed design phase which covers:

The definition of the agent model specifies the system agent classes and their instances. GAIA allows for one-to-one mapping between roles and agent types as well as many-to-one mapping between roles and agent types which can lead to more efficiency.

The definition of the services model identifies all required services or activities that the system agent would need to play their roles and to demonstrate their properties.

The GAIA aims at offering a method centered around organisational abstraction and it has succeeded in delivering a clear method that smoothly takes the system designer in a sequential theoretical journey, starting from analysis, to design to implementation. The use of textual templates for agent roles is recommended, and an approach of defining one per role helps inform the protocols, permissions, and responsibilities, in terms of liveness and safety, as well as the overall description for each agent role [Henderson-Sellers, 2013].

GAIA analysis gives more attention tp late requirements of system development and assumes a complete set of requirements is already gathered, however the authors have not rejected the idea of integrating an early requirements analysis stage. They admit that GAIA suffers limitations caused by the incompleteness of its set of abstractions and the missing holistic

view of the environment. The focus on the role model means global organisational rules are overlooked or missed, which makes it unsuitable for modelling open systems or for controlling systems where agents are all self-interested. This issue however is dealt with in the work of Juan et al. [2002]. In their proposed extension "ROADMAP", they allow for better engineering of open systems through: (i) supporting requirements gathering, (ii) adding explicit models to describe the domain knowledge, and environment, (iii) adding explicit models to describe the social aspects and individual aspects of the agents. Other drawbacks of GAIA are that the methodology does not directly deal with implementation issues and it does not explicitly deal with the early activities of capturing requirements at early stage. We contend that GAIA and its extension ROADMAP are a very lengthy and complex processes and that they also lack the formal presentation of their concepts, which makes it hard to verify the system model and makes it hard to use for non-specialists. Its lack of an implementation phase makes it even harder to map its concepts to actual code constructs.

2.3.2 Prometheus Methodology

With the aim of making MAS accessible for a wider spectrum of users and offering a practical approach for developing MASs, Padgham and Winikoff [2002] developed the Prometheus Methodology. Prometheus defines a detailed process to specify, implement and test/debug[1] MASs. The core distinguishing features of Prometheus are:

1. Prometheus provides detailed guidance on its process and on each single step of that process.
2. It supports the development of goal and plan based agents.
3. It covers a wide range of activities that take the process from requirements gathering to the creation of a detailed design.
4. It is supported by a tool (PDT)[2] that is freely available for users.
5. It is aimed at industrial software developers and undergraduate students with no or little experience of MAS paradigm.
6. It has been developed and enhanced based on input from students as well as commercial collaborators.

The Prometheus methodology consists of three phases each contains a number of graphical models, that reflect the structure or the components of the system and textual descriptor forms that provide the details needed for each component. Both the models and descriptors aim at covering all system dynamics to form a comprehensive model.

1. **System Specification:** In this phase, the user defines the system at a high level by defining *system goals*, *use case scenarios* and *basic system functionalities*. Also how the system interfaces with its environment through defining what *actions* it does and which

[1] Early attempts were done by Poutakidis et al. [2003]

[2] Prometheus Design Tool (PDT); `http://www.cs.rmit.edu.au/agents/pdt/`, retrieved 20 January 2014

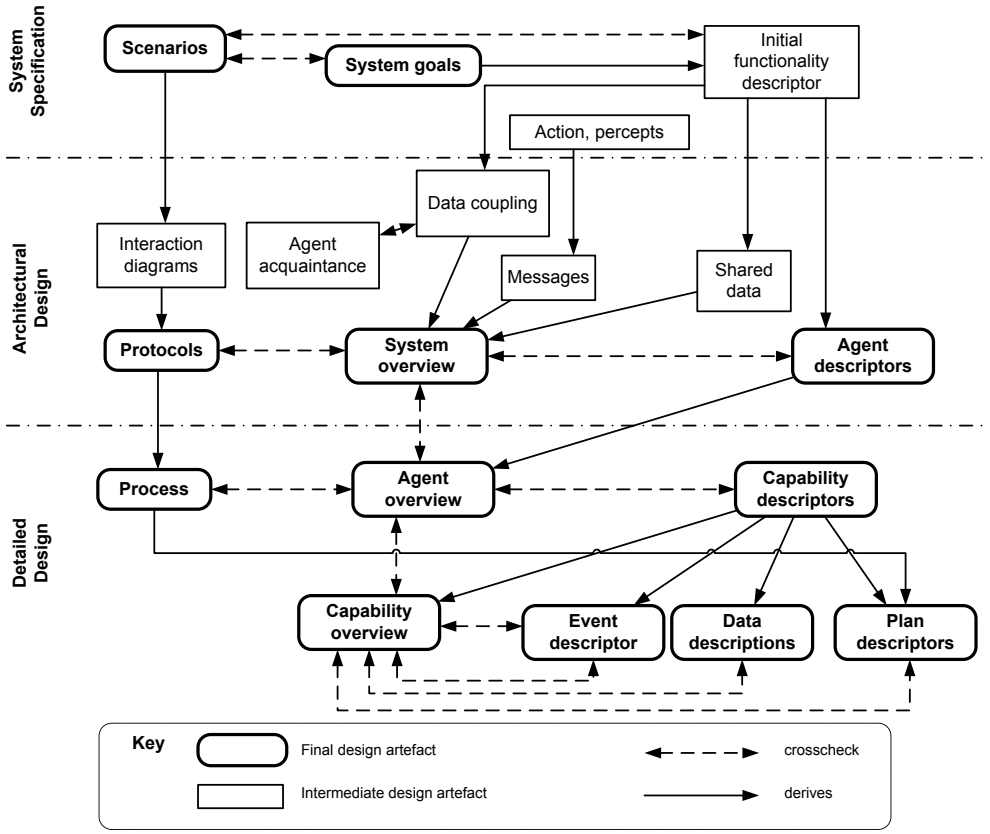


Figure 2-5: *Prometheus Methodology Overview (used with permission) [Winikoff and Padgham, 2004]*

percepts it senses. The user can start by defining the initial system goals and their subgoals, then grouping each set of relevant goals together to define the system functionalities, this includes identifying actions, percepts and data interchange. Following that the user can describe the different use case scenarios, which are sequences of events associated with achieving a particular system goal. Finally a definition of the environment which the agent will be situated within, by describing the percepts and actions as well as any external data or code available for the agents.

2. **Architectural Design:** This phase includes deciding on the *agent types* by grouping functionalities though a coupling process developing the *agent descriptors*. The defining of the system dynamics in terms of *agents' interaction protocols* through interaction diagrams, which are normally derived from the use case scenarios. And finally, capturing the *overall system structure* through the system overview diagram which shows the agents' types, their actions and percepts and both internal and external data they access. The system overview diagram is important as the first entry point to understand the system architecture, the visual components in this diagram have distinctive visual depiction and they are linked by directed arrows that indicate the sender and recipient of messages, which actions are performed by which agent, which percepts are received by

which agent, and what data is read and written by which agent.

3. **Detailed Design:** this phase focuses on developing the internal design and details of the system agents – through the creation of agent overview diagram – by deciding on which *capabilities* each agent has and defining the *plans* and the triggering *events* associated with these plans. Furthermore the development of process diagrams to describe the interaction protocols that present the local view for each individual agent. Finally the refining of the capabilities details through the capability overview diagrams and various descriptors where the system designer can develop plan sets to achieve the agents/system goals, and identify which events trigger the execution of these plans.

In its time, Prometheus appeared to be complete and mature among other existing MAS development methodologies. It has also proved its practicality, to some extent, through its application in both industrial and academic environments. The methodology comes with the Prometheus Design Tool (PDT) that supports the full process (system specification, architectural design and detailed design) and there are some guidelines on how to convert the system design to JACK Development Environment (JDE), that is able to generate a skeleton code for JACK [Busetta et al., 1999].

On the other hand, Prometheus has been criticised for its simple presentation of goals that is done by using a simple version of KAOS[3] [Dardenne et al., 1993] without the inclusion of business model descriptions. An attempt to address this issue motivated the the work of Cysneiros and Zisman [2004] where they propose the use of the i * technique [Yu, 2011] to allow for modelling both goals and business processes, however, this work was not integrated into Prometheus. Another identified issue with Prometheus is, besides the lack of automated code generation, that Prometheus models cannot be fragmented, do not include deployment diagrams and has no way to simulate the system to check before generating the final code. The identification of these issues led Gascueña and Fernández-Caballero [2009] to propose to combine Prometheus and INGENIAS [Pavón and Gómez-Sanz, 2003]. This work suggested using Prometheus to create the initial design then move that to INGENIAS to complete the modelling. Similarly, Sokolova and Fernandez-Caballero [2010] proposed combining Protégé with Prometheus to support a full life cycle for designing MASs. These attempts generally establish that Prometheus, although aimed at being a complete methodology for developing MAS, it is not offer enough to support various use cases or the ongoing advances in MAS development.

2.3.3 The TROPOS Methodology

TROPOS [Bresciani et al., 2004] distinguishes itself from other methodologies by paying great attention to the requirements analysis. The modelling process consists of five phases that cover analysis and design and it uses JACK for the implementation, the developers however need to map their design concepts into JACK's [Busetta et al., 1999] five constructs. TROPOS offer

[3] KAOS stands for Knowledge Acquisition in autOmated Specification

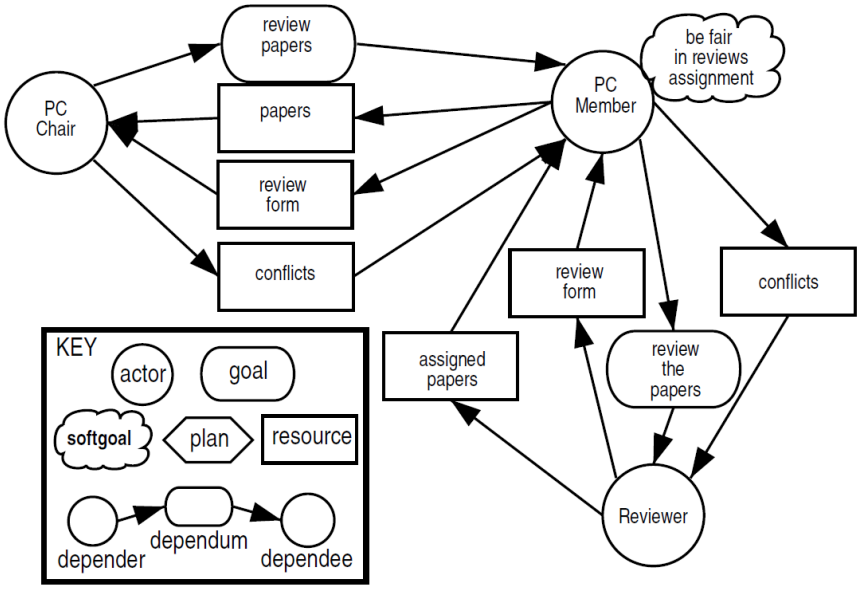


Figure 2-6: *An Example TROPOS Actor Diagram (used with permission) [Susi et al., 2005]*

some guidelines to help in this process, but it seems very lengthy and complex.

TROPOS adopts a requirements driven software development approach, by exploiting goal analysis and actor dependency analysis techniques founded on i* [Yu, 2009]. TROPOS uses small set of knowledge-level notions across all phases and it presented in its time two novel features:

Agent Notion and its BDI-based mentalistic notions are prominent throughout the whole process from analysis to the actual implementation.

Early requirements Analysis as part of the methodology – a distinguishing factor of TROPOS compared to other existing methodologies.

The modelling process include the following activities:

1. **Actor Modelling**: To identify and analyze both the actors and their intentions as well as the data flow and controls in the environment.
2. **Dependency Modelling**: To identify which actors depend on each other, the plans to achieve the goals and finally the resources to be used/consumed.
3. **Goal Modelling**: To identify from the actor point of view the main goals and their sub-goals which is done using three different techniques *means-end analysis*, *contribution analysis*, and *AND/OR decomposition*.
4. **Plan Modelling**: To identify the plans and their sub-plans to achieve the identified goals – which is done using the same techniques as for goal planning.
5. **Capability Modelling**: To identify which goals/plans the actor needs to be able to define, choose and execute a plan or to maintain its social ability/relation to other actors or system components.

TROPOS's five phases are:

1. **Early Requirements Analysis:** the focus of this phase is to identify and analyze the stakeholder (as social actors) and their intentions. Modelling intentions as goals is through goals-oriented analysis to decompose goals into finer ones, then using means-end analysis to identify plans, resources and soft goals. In this phase the user creates the actor diagram, and goal diagram.
2. **Late Requirements Analysis:** is about defining the system functions in terms of functional and non-functional requirements considering the system as one actor and examining its dependencies on other organisations in the environment.
3. **Architectural Design:** is to define the system's global architecture through the definition of the sub-systems (actors) interconnecting through data and control flows (dependencies). This normally done in three steps, the first is to define the overall architectural organisation, the second is to identify the capabilities needed by the actors to fulfill their goals and the third is to define a set of agent types and assign to each of them one or more different capabilities.
4. **Detailed Design:** is to define the detailed specification of the agents at the micro level, and how they communicate. This step is strongly dependent on the implementation choice. The detailed design phase takes the architectural specifications and tracks back to the reasons for a given element as early as the requirement analysis
5. **Implementation Using JACK:** the system elements identified are mapped to JACK constructs.

The TROPOS methodology's inclusion of organisational issues, with explicit study of its structure from the earlier stage, emphasises how important the organisational structure is for the development of agent systems. The TROPOS methodology defines coherent guidelines for all of its activities, which makes it easy to follow, however TROPOS's analysis lacks identification of organisational rules. There is no single model that is able to capture the global constraints that might govern the system actors or apply to multiple organisations. The side effect of this is that it shifts the modelling effort more from the analysis phase towards the design phase, which might over-complicate the latter. Although TROPOS was proposed without formal specification, Fuxman et al. [2003, 2001] have suggested Formal TROPOS that extends the TROPOS methodology with formal specification by integrating TROPOS primitive concepts with a temporal specification language. Formal TROPOS employs model checking verification techniques and had a prototype tool (T-Tool) that supports the approach. A further extension of TROPOS is the work of Mouratidis et al. [2002] which included more concepts to allow for the support of security requirements during analysis and design time. B-TROPOS [Bryl et al., 2008b] is more recent extension work of TROPOS in the form of combining business goals and requirements with business process modelling. In this work an inclusion of declarative business process-oriented constructs, inspired by the DecSerFlow and ConDec languages, that allows for great flexibility in terms of modelling time constraints and other forms of system constraints. B-TROPOS can be mapped to $\mathcal{S}$CIFF for properties and conformance verification

purposes.

2.3.4 SONIA Methodology

According to Alonso et al. [2005], other agents software development methodologies are unable to provide a *natural* process for modelling Agent Societies or MAS.In consequence, the authors identify the following features as essential for a MAS development methodology:

1. MAS methodology should not have the presumption of using agent paradigm as best design option from the start of the analysis phase, instead the analysis should be fully independent from technology and only after analysis is complete the decision either to use agent paradigm or not can be made.
2. It should naturally lead the system design to that conclusion to use MAS or not, at the moment most other methodologies depend on the system designer's experience to decide.
3. It should lead the identification of MAS components in a systematic way, rather than depending on the system designer's experience in doing so.
4. It should naturally help in creating the system organisational model. While other methodologies focus on only the agent internal architecture and its interactions with other agents, the methodology should cover the social organisation concept.
5. It should help in creating reusable agents, as the concept or reusability is core in all modern software development engineering and should be preserved in the agent development field.

SONIA covers only two phases; *Analysis* and *Design*, each phase is divided into two stages totalling 11 different models. Below is a summary of each of these phases:

1. **Analysis:** Analysis is done in two stages, the first is ***Conceptualisation*** stage; which is high level analysis of the problem domain structure and which identifies general tasks to solve these domain problems. This stage uses the method of Set Theory Based Conceptual Model (SETCM) Alonso et al. [2005].This method is used mainly because it is design-independent, which satisfies the first methodology requirement. During this stage two models are created: (i) the *Initial Structure Model* that describes the general structure of the problem domain and (ii) the *Initial Task Model* that describes how these problems can be solved. The second stage is ***Extended Analysis***, where the previous models are redefined and expanded to capture the system environment and any external entities. In this stage there are three models: (i) *Environment Model* that defines the external entities to the system and how they interact with it (ii) *Structural Model* that defines the structures from the external entities that interacts with the system, and (iii) *Task Model* which adds the required functionalities to facilitate the interaction between external entities and the system.
2. **Design:** phase is done in two stages. The first is ***Synthesis***: grouping the elements of both structural and task models based on relevancy of concepts, such as knowledge,

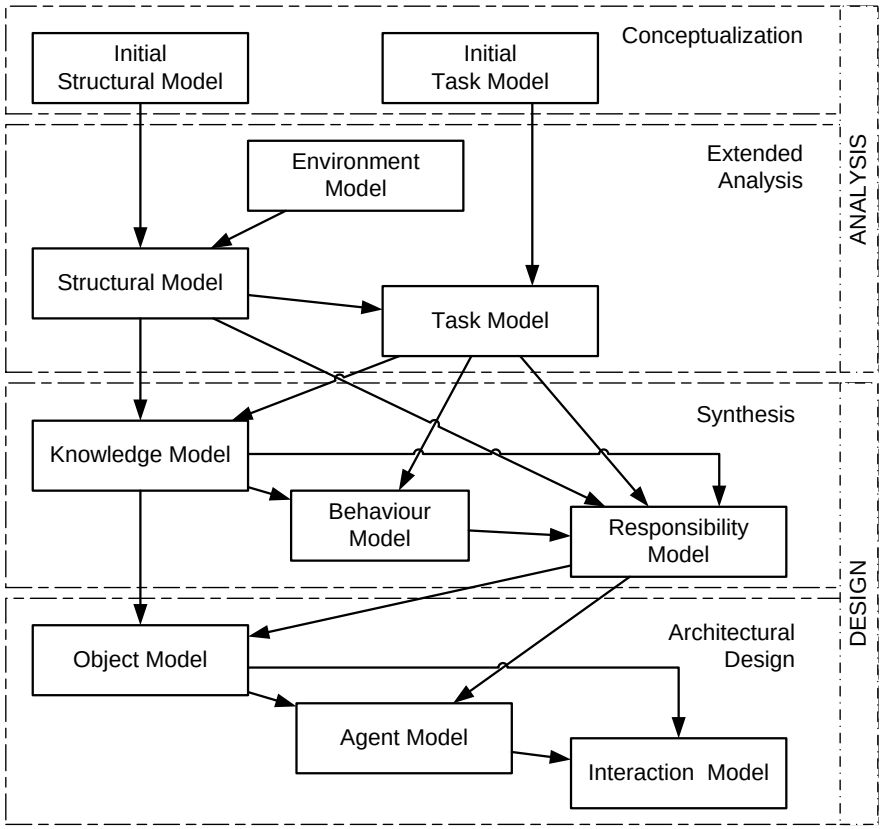


Figure 2-7: *SONIA Methodology Overview (used with permission) [Alonso et al., 2005]*

behaviour responsibility. During this stage three models are created: (i) *Knowledge Model* that identifies the knowledge components by grouping structural model concepts and association (ii) *Behaviour Model* that identifies agents behaviours and is created by grouping the task model tasks, sub-tasks and methods, and (iii) *Responsibility Model* that maps related knowledge components to behaviours. The second stage in the design phase is ***Architectural Design***, which focuses on the identification of architecture components through the creation of three models: (i) *Agent Model* that identifies and defines using the behaviour model components what entities should be designed as agents (ii) *Object Model* that identifies and defines using the behaviour model the reactive components that needs to be shared between agents or other system components, and (iii) *Interaction Model* that identifies and defines the required Agent/Agent and Agent/Object relationships.

Evaluating SONIA based on the goals set by its authors marks it out as very good MAS methodology: it has met the requirements they have set in terms of making the analysis phase independent of MAS as the ultimate solution, also the design process appears seamless and natural. SONIA, however, does not cover the full life cycle of development: there are no given guidelines regarding how to implement the designed system. Modelling the tasks and the creation of the task model are considered at early stage of SONIA while the goals are actually

identified at later stage: this contradicts the rational design cycle where tasks and actions are considered as means to achieve the identified goals [Henderson-Sellers, 2013]. SONIA considers the dynamic components as agents but the concept of actors/human components is not considered. Also the view of the static components is not clear whether they are dynamic services that act in a reactive manner, or just shared resources for all dynamic components to use. These issues expose SONIA as a potentially insufficient methodology, although there are points to learn from it specially regarding the need to justify the suitability of MAS for the system to be modelled.

2.3.5 AGR: Agent/Group/Role for Organisation Centered MAS

AGR [Ferber et al., 2004] is a basic methodology that was developed with a primary focus on supporting the organisational structure of MAS. The first metamodel was proposed in 1997 by Ferber and Gutknecht [1997], which was a meta-model for describing hierarchical organisations[4]. AGR has a minimal metamodel around just three concepts (Agent, Group and Role), although later some work was done to include the environment within the model, such as by Odell et al. [2003], that included the environment and associated it to the group, also the work of Ferber et al. [2005] that proposed AGRE where the environment was included and modelling the relationship between the agents and their environment by introducing the concept of body and mind. In this section we summarise AGR and highlight its more recent replacement MASQ [Stratulat et al., 2009]. There are three primitive concepts that form the core of AGR:

1. **Agent:** that is an active, communicating entity that plays one or more roles within one of more groups. There is no constraints upon the architecture of an agent or any restrictions of its its mental capabilities.
2. **Group:** a group is a set of number of agents that share common characteristic, any two agents may communicate if and only if they belong to the same group, although an agent may belong to more than one group, hence in theory that agent could communicate to any member of any group it belongs to.
3. **Role:** "a role is an abstract representation of a functional position of an agent in a group" [Ferber et al., 2004]. An agent must play at least one role in a group, and it might play several roles in one group or across many groups. A number of agents may play the same role in the same group.

AGR allows for adding structural constraints to describe the relationship between roles. AGR allows two types of constraints; the first one is ***correspondence***: the relation between ***role A*** and ***role B*** means that an agent that is playing ***role B*** will automatically play ***role A***. The second type of constraints is ***dependence***: where it defines the dependencies between group membership and role playing, in other words the group member needs to play one role before being able to play another role (e.g. a lab director has to be researcher as well).

[4]Previously named Aalaadin

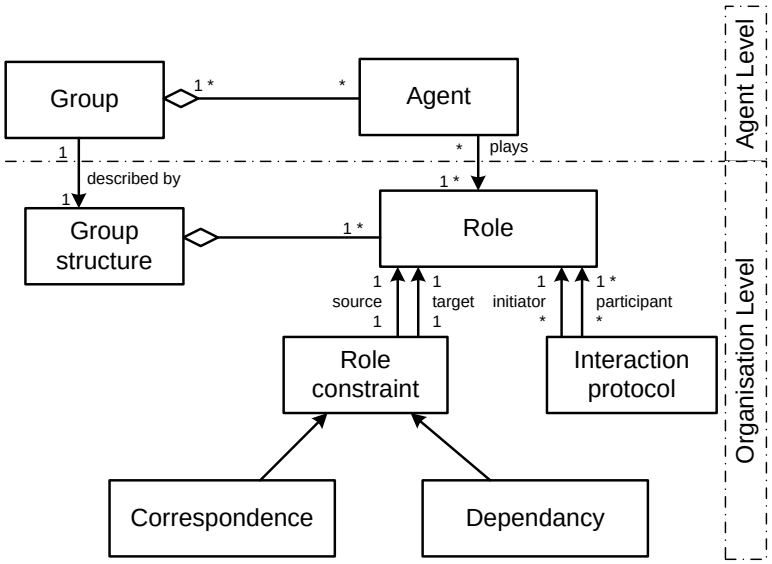


Figure 2-8: *The UML metamodel of AGR (used with permission) [Ferber et al., 2004]*

Ferber et al. [2004] present three different types of visual modelling diagrams to describe a MAS:

1. ***The organisational structure diagram:*** this is used to describe the organisation and its relationships at abstract level. In this diagram groups are presented as rectangles, roles presented as hexagons and located in the group rectangles. Constraints are represented as connecting arrows between the different roles, where correspondence is represented as large arrows and dependencies presented as thin arrows. Finally, they use rounded rectangles to present the communication protocols between different roles, the initiator role is represented by an arrow that points towards the interaction.
2. ***The organisational sequence diagram:*** this is a variant of the sequence diagram of AUML [Bauer et al., 2001]. It is used to describe various dynamics of the organisations including the temporal relation between organisational events such as the creation of groups and agents entering and leaving a group as well as the acquisition of a role. The life cycle of an agent is represented by several segments of the same colour and distributed between the different roles that are grouped.
3. ***The group adhesion process cheeseboard diagram:*** In this diagram a group is represented as an oval (cheeseboard), while agents are represented as skittles that stand on the board and go through the board if they belong to several groups, finally a role is presented as a hexagon where each agent plays this role is linked to the hexagon that represent that role.

AGR model is so simple as it includes only three concepts. This allows for describing the organisation structure at an abstract level and does succeed in avoiding the agent-centric view that other methodologies have a strong bias towards. Although Ferber et al. [2004] proposes a notation for modelling Organisation Centered Multi-agent Systems (OCMAS) they have not

defined a full methodology for the system developers to follow, they just highlight the key point of how a methodology could be defined based on an OCMAS model. They propose to use the GAIA methodology to fill the roles and relate them to the general structure. The lack of clear steps to model a system, we contend, is the biggest drawback of this methodology. Furthermore, assuming the highlighted key points are sufficient, AGR gives great attention to the organisation structure and ignores the other essential details of defining an agent and leaves the area of modelling other possible components of the system wide open. The advances in current distributed systems take into account other system participants such as services and human actors. AGR does not cover this view and focus on the basic definition of MAS as collection of only agents. "AGR argues that agents can have their joint behaviour orchestrated by interaction protocols, but the nature and the primitives to describe such protocols are left open" [Isern et al., 2011]. MadKit[5] is a Java platform that support AGR and can be used for practical implementations. Neither AGR nor AGRE integrate the system norms and institution structure, because their models are based on OCMAS principles that are concerned with the mental state of the agents [Ferber et al., 2009].This issue has been acknowledged by the authors and led to proposing new extension in the form of new framework called Multi-Agent Systems based on Quadrants (MASQ) [Ferber et al., 2009]. The MASQ framework is based on 7 concepts and its metamodel contains five basic concepts: mind, object, bodies, brute space and cultures as well as a set of relations between these concepts and a set of laws that describe the system dynamics. Although MASQ's novel approach solves many of the issues in both AGR and AGRE, it increases the modelling complexity of the system and it is definitely not easily understood by less experienced system designers, which makes it less accessible.

2.4 Review of a selection of Metamodels for MAS

Early literature on agents and multi-agent systems described the agent concept and its abstract architectures by giving formal descriptions. The work of Wooldridge et al. [2000a] was somewhat limited in terms of exploring all aspects of multi agents as social entities, while the work of d'Inverno and Luck [2004] was very detailed however, it did not present a pure abstract view of the agent. Rather the agent and multiagent view was heavily influenced by the system objects and directed towards defining the relations between agents as collaborative entities. More recently a wealth of metamodels have emerged to support the development of Agent Oriented Systems, such as the UML-based metamodel specifications of ADELFE, Gaia and PASSI [Bernon et al., 2005], the MASQ metamodel [Dinu et al., 2010] and the work of Omicini et al. [2008]. We cite these for the benefit of the reader to find more details on each of them, however we limit our exploration of the related work to only three of the recent and most promising metamodels; two of them are an attempt to define a generic metamodel for MAS and the third is a recent attempt to define a metamodel that is more focused and directed

[5] `http://www.madkit.org/` retrieved 20 January 2014

towards the the support of agent-based simulations.

2.4.1 A Platform Independent Metamodel for Multiagent Systems

Hahn et al. [2009] examine various multiagent metamodels including Aalaadin, ADELFE, Gaia and PASSI, then propose a unified MAS metamodel by merging the metamodels of ADELFE, Gaia and PASSI to cover all of their aspects.

Hahn et al.s' unified metamodel for MAS adopts multiple points of view to cover all the features in the different technologies. These views we now summarise:

Multiagent View: the main blocks of any MAS; and covers agents, their capabilities and the primary concepts of any MAS such as cooperation, interaction and environment.

Agent View: describes each individual agent and the capabilities it uses to achieve its tasks, as well as the roles it plays in the context of MAS.

Behavioural View: plan composition, specification of how atomic tasks are done and how data flows within system control constructs.

Organisation View: the organisation structure and how cooperation is achieved between the system and individual autonomous entities.

Role View: identifies the functional states of the system's autonomous entities and their social relationships.

Interaction View: how autonomous entities interact and the form of this interaction.

Environment View: the different kinds of resources that are created by the agents or shared by the organisations.

Hahn et al. propose a complete model, called PIM4Agents, that defines the abstract syntax of a domain-specific modelling language for MAS (DSML4MAS), as well as a definition of the model transformations from Platform-Independent Model (PIM) to Platform-Specific Models (PSM), which allows for the transformation of the designed model into an implementation for a specific platform such as JADE or JACK.

PIM4Agents is truely a generic approach and a comprehensive description of MAS based systems. This comprehensive metamodel is helping in understanding the wide range of domain specific concepts that could be used in wide range of MASs. However, Hahn et al.'s decision to combine all concepts of the chosen three metamodels leads to an increase in complexity of their model and modelling MAS systems using these set of concepts becomes more difficult. Although a unified metamodel can help in moving MAS towards a standard form, it cannot satisfy some situations where a specific focused structural view of MAS is needed.

2.4.2 FAML: A Generic Metamodel for MAS Development

Motivated by the advances in Model-Driven Development [Atkinson and Kühne, 2003] Beydoun et al. [2009] committed to the mission of combining all different metamodels in the

domain of MAS to produce one generic metamodel. Beydoun et al. attempt to develop a unified metamodel to allow for interoperability, better understanding and better communications between researchers. FAML metmodel was created following a four-step process:

1. Determination of the full set of general concepts relevant to any MAS and its model.
2. Short-listing the candidate definitions.
3. Reconciliation of definition differences to build a consistent set of metamodel terms.
4. Designation of the chosen concepts into two sets: design-time and runtime, where the central design-time concept is system as an agent-oriented system while the central runtime concept is the environment wherein agents reside.

In FAML, an agent has internal and external concepts, and those classes that relate to the agent's internals at design-time are called *agent definition level*, while those relate to the agents internal aspects at run-time are called *agent level*.

Classes that relate to the agent's external aspects at design-time are called *system level*, while classes that relate to agent external aspects at runtime are called *environment level*. In summary:

Design-Time Aspects: the system has an agent-oriented structure that satisfies both functional and non-functional requirements. Roles are also used to describe the system. They are normally related to tasks, either as responsible for a task or as a collaborator in a task.

Runtime Aspects: The environment is an essential part of the system: it is where the agents reside and it provides the facets they need to interact.

In FAML, the environment has a history which is a composition of all message events and facet events that occurred in the environment. Agent internals at runtime comprise the collection of beliefs, desires, and intentions an agent can hold, including support for basic BDI concepts, but these are not compulsory. Finally the actions that make an agent plan can be facet actions or message actions.

FAML offers a generic approach to the description of any MAS, and having a comprehensive metamodel that fits all views and approaches of developing MAS is a great contribution that has helped us to verify that our concepts could be mapped to other metamodels. However, we find that the inclusion of so many concepts in FAML has led to increased complexity and introduced a steep learning curve to be faced by any new entrant in the field of MAS.

2.4.3 AMASON: Abstract Meta-model for Agent-based SimulatiON

AMASON [Klügl and Davidsson, 2013] is a metamodel proposed to support the building of multiagent based simulation systems. These systems are somewhat different to modelling a real life application. Despite the wider recognition of the importance of metamodelling, few attempts were made to develop a metamodel for simulations the equal of those proposed for MAS. Two recent proposals are MAIA [Ghorbani et al., 2013] and easyABM [Garro and Russo, 2010] which provide a social-focused language for describing agent-based simulation

systems. Neither of them was preferred by AMASON's authors, as both are described as having a detailed view and share the many problems of the traditional AOSE approaches. Hence AMASON is proposed as a better alternative that provides a basic view, using a minimal set of aspects, and being suitable for all types of models.

AMASON has two basic views:

- The context of the model: which contains information about the model objectives, how can it be used, and under which parameters etc.
- The content of the model: which is the data of the model and its actual objects.

AMASON Model Elements

AMASON has three types of components for a multi-agent simulation model: Body, Mind and Regions. Each of these could have at its internal state, the separation between the physical entity from the mental one, this is done mainly to address the need for a clear distinction between entities that possess reasoning capabilities (agents) and those that do not (services). In this section we examine each one of these, as shown in Figure 2-9.

1. **Body:** a body is the presentation of the physical entity in a Multi-Agent Based Simulation (MABS) model such as a human, or the physical parts of a robot, rocks, food items ... etc. The body has a domain-specific state and is normally located in specific region. The body state can be updated by region-specific processes.
2. **Mind:** is the mental state of the body, so for a body to become an intelligent system component (agent) it needs to be coupled to a mind. The mind normally contains the reasoning capabilities that allows it to handle decision-making processes. The internal state of the mind could have memory or a reasoning mechanism based structure, depending on its design. The coupling between the body and the mind is called embody in AMASON.
3. **Region:** is the spatial environment where the agents and objects are situated. In all cases, except where all agents are virtual, an explicit structure of the environment is required. In AMASON a region is conceived as an explicit entity with a state so its global properties can be captured similarly to the body. AMASON supports the concept of a region independent entity as well as hierarchies where a region is a container located within another region. The structure of the environment takes the form of interconnected regions, which originates from a generalisation of spatial presentations observed in various MABS models [Klügl and Davidsson, 2013]. Due to the complexity of the interconnected regions, having this concept subsumes presentations ranging from a single region with a grid map to a network of regions and it allows formulating environmental heterogeneity beyond heterogeneous populations.

AMASON offers clear concepts for modelling agent based simulations, and it has succeeded in keeping the agent concept as simple as possible and in presenting an abstract view of the system components from data types. Additionally the idea of connected regions is very

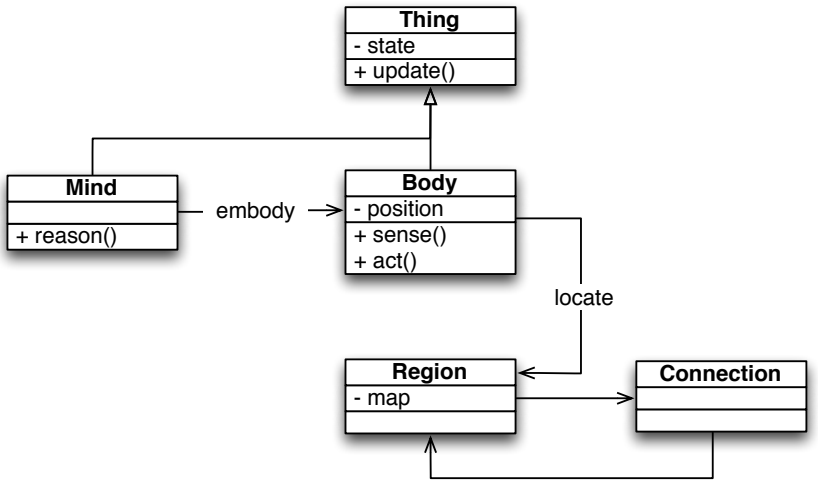


Figure 2-9: *AMASON Metamodel: Overview of AMASON's Elements*

interesting and is useful in solving relations when dealing with MAS organisational structure. The authors of AMASON admit that their work is still initial and they expect to develop it more to support social aspects, which have been ignored for the time being. The current proposal covers only the basic concepts, it does not define data types or other details which push more work on the designers and developers and might lead to confusion and various extensions to address the different needs that might be different from one group to another. AMASON does not provide any language or support for interactions, coordination or organisation and its current state might only suit limited number of use cases.

2.5 Business Processes Management

Business Process Management (BPM) is not a new field. Consider ancient civilisations, such as in Egypt: no one can imagine the creation of complex temples and the famous pyramids was done without proper management of all the various engineering processes and tasks involved. Modern scientific management, however believed to be associated with Fredrick Taylor's theory of management dates back to 1911. Taylor laid the first foundations of systematic observation and study that influenced how scientific work is managed for over seventy five years [Sinur et al., 2013]. Applying management science to the *process* came however later through the work in total quality management of Juran and Gryna [1970]. Their work is considered to be the first wave of business process management. Nine years later notable development of how processes should be managed started with the arrival of Enterprise Resource Planning and Business Process Re-engineering. As a result, many techniques have been developed, contributing to the second wave of process management. The third wave followed in the early 2000s when Smith and Fingar [2003] highlighted the need for putting IT into process management. Greater control of information systems meant more agile processes and optimised

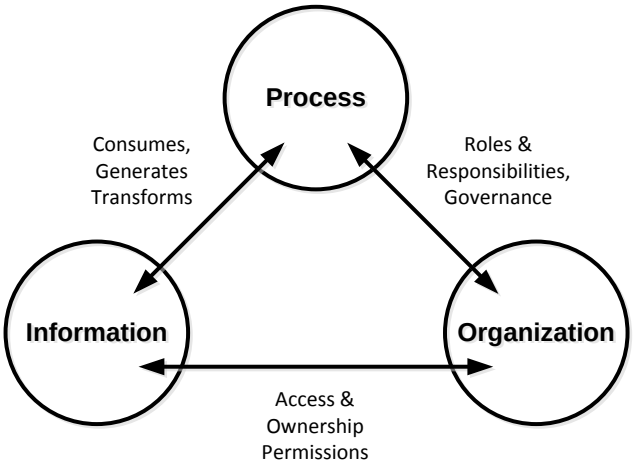


Figure 2-10: *Relationship between Information, Process and organisation (used with permission) [Hollingsworth et al., 2004]*

execution in the form of IT-driven business processes. BPM by then became associated with modelling and analyzing and documenting the processes of design and execution, as well as the arrival of automated business process discovery. The definition of a business process by Hammer and Champy [1993] as "a collection of activities that take one or more kinds of input and create an output that is of value to the customer", has influenced the development that followed.

Jennings et al. [2000] have recognised the need for business processes to become more intelligent and proposed the integration of agents into business process management. In their work, Advanced Decision Environment for Process Tasks (ADEPT), they conceptualised, designed, and implemented the business process management system using an agent-based approach. In their implementation, the business process is designed as a collection of autonomous problem-solving entities, and are able to negotiate with one another to reach mutually acceptable agreements. This approach means that agent based business process systems could offer a greater degree of flexibility, agility, and adaptability over traditional systems.

In this section we highlight some basic concepts of BPM and introduce the concept of Smart Business processes that shows how the use of agent technology can transform BPM for the better. Reading this section is essential to understanding the relationship between BPM and MSMAS concepts.

2.5.1 BPM: Conceptual Model and Terminology

The model of the concepts at the core of business process management as shown in Figure 2-14 [6] defines business processes as a collection of activities whose coordinated execution realises some business goal [Weske, 2012]. The activities can range from being system activities, user interaction activities, to manual activities. Although manual activities are not supported

[6] expressed in the Unified Modeling Language, an object-oriented modelling and design language

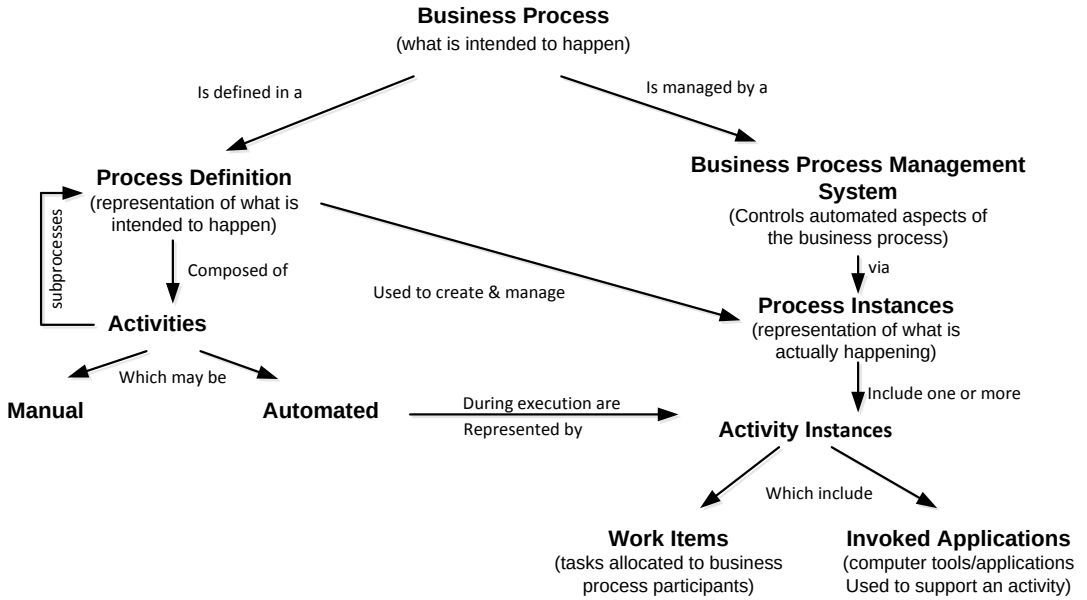


Figure 2-11: *Constituent Components of a Business Processes (used with permission) ([Hollingsworth, 1995])*

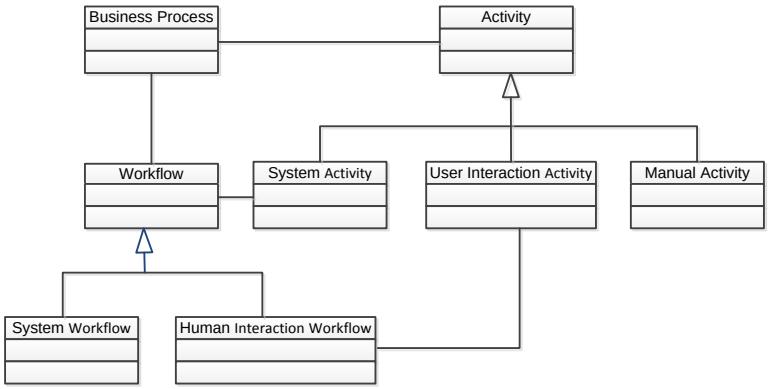


Figure 2-12: *Business Process Conceptual Model (used with permission) ([Weske, 2012])*

by information systems, they are part of the modelling process, especially if the human user activities require an interaction with the information system, e.g. filling in a form describing or confirming the physical activity execution. Parts of the business process can be enacted by workflow technology where a sub-system is making sure that the activities of a business process are performed in the order specified, and that the information systems are functioning in a way that leads to the realisation of the associated business goals.

2.5.2 BPM: Abstraction Concepts

To capture the complexity in business process management, Weske [2012] has defined two different abstraction concepts. The first is the *Horizontal Abstraction* that is based on the traditions of computer science to separate the modelling levels, as shown in Figure 2-12. This

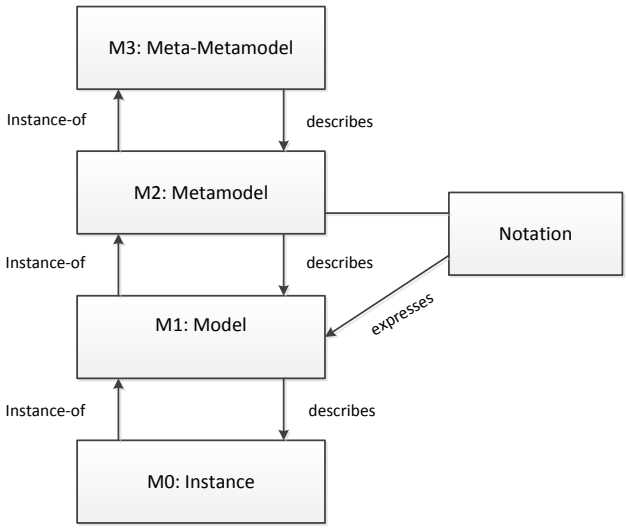


Figure 2-13: *Horizontal Abstraction Business Process Conceptual Model (used with permission) ([Weske, 2012])*

abstraction follows the levels of abstractions identified by the Object Management Group [7], the metamodel level, the model level, and the instance level are important for the design and analysis of software systems. The instance level reflects the concrete entities that are involved in business processes such as activities, concrete data values, resources and persons. The model level is to describe the business process scenarios and it identifies and classifies various entities used at the instance level as well as the organisation of the system. Models are expressed in terms of metamodels that are associated with notations of a graphical nature. The entity relationship metamodel also defines entity types, relationship types, and connections between them. Finally the metamodel is described by a meta-metamodel that allows for transformation and mapping from a modelling technique to another.

The second abstraction concept is *Vertical Abstraction* where an identification of the modelling efforts is presented as sub domains of process modelling, as shown in Figure 2-13, in which there are four sub domains. The first sub domain is *Functional Modelling*, which investigates the units of work that are being enacted in the context of business processes. Specifying these functions can be done informally using English text or formally using syntactic or semantic specifications. The second sub domain is *Information Modelling*, where an investigation of the data involved in business processes is done: identification of data values and dependencies between business activities has to be taken into account when designing the business process to be able to handle issues raised if a function requires certain data type that is not available at execution time. The third sub domain is *Organisation Modelling*, where business activities can be associated with particular roles or departments in the business organisation. The identification of roles is essential to control and specify who is responsible for each activity. The fourth sub domain is *Information Technology Landscape Modelling*, that defines which information sys-

[7] Object Management Group (OMG): `http://www.omg.org/`

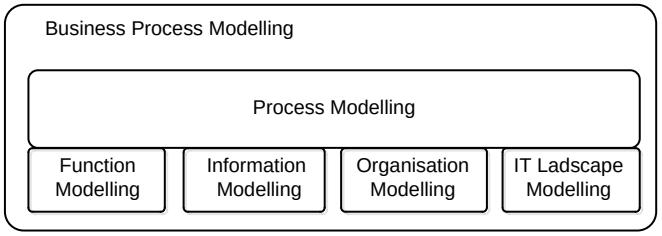


Figure 2-14: *Vertical Abstraction of Business Process Conceptual Model (used with permission) ([Weske, 2012])*

tems assist and support the execution of various business activities: this domain takes care of identifying which information systems, their relationships, and their programming interfaces, are needed to be present to be used for each function. *Process Modelling* defines the glue between these four sub domains and a process model links the functions of a business process with any execution constraints to specify ordering and conditional execution requirements.

2.5.3 Modelling Business Processes Interactions

Enterprise cooperation is becoming common practice, also intra-organisation interactions are common at the department level. As a result, when modelling business processes we should consider how different business processes, that reside in different organisations or different departments, interact. These interactions typically occur in a peer-to-peer style, according to predefined process choreography. Figure 2-15 shows an example of interacting processes, where a buyer orders some products from a reseller: each enterprise is represented as a value chain [8] and within these value chains are the business functions that are realised by the business processes. The buyer's *place order* business activity interacts with the reseller's *receive order* activity by sending a message. In the reseller side, this message is received by a *receive order* activity and the process continues as specified. Interacting process instances in BPM are visualised as *Event Diagrams*, where a horizontal line is used to present each participant, on which the events of that participant appear in an ordered fashion. Participants communicate by sending and receiving messages. Figure 2-16 shows the event diagram of the sample interacting processes shown in Figure 2-15.

2.5.4 Modelling Organisation

An essential part of BPM is the coordination between the personnel of an enterprise. This is normally done by modelling the organisational structures within the enterprise in which the business process will execute. The concepts used to describe the organisations are positions, roles, teams, and relationships between positions. Behind the organisation model is the resource which is an entity that can perform work for the enterprise, such as humans and other

[8] A Value Chain is a high-level model of how businesses receive raw inputs, add value to them by conducting various processes leading to a finished product or service.

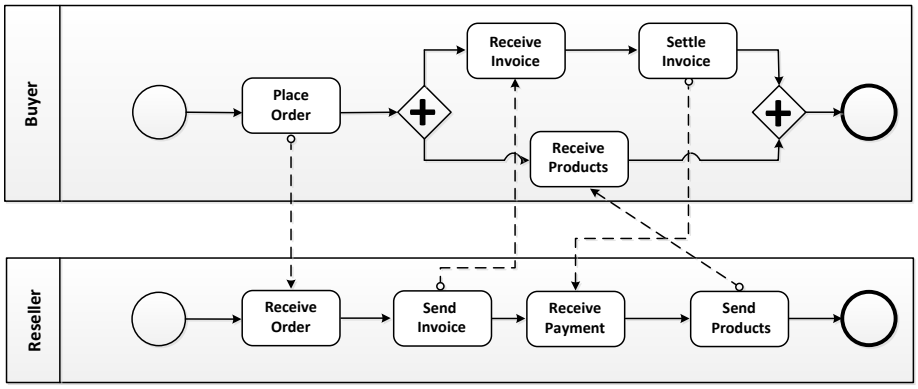


Figure 2-15: *Business Processes Interaction Model (used with permission) ([Weske, 2012])*

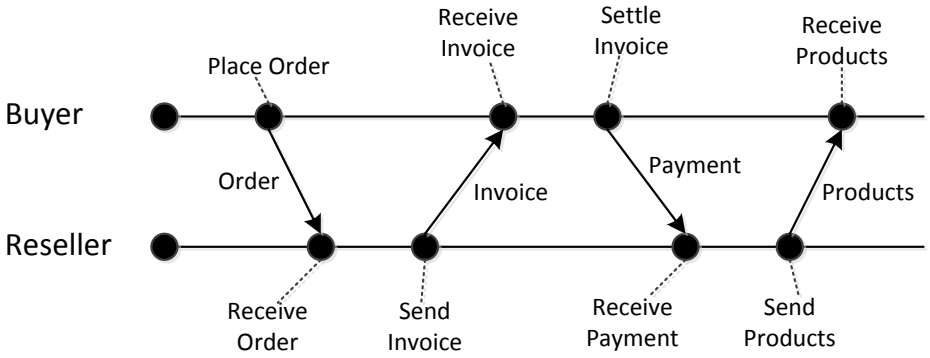


Figure 2-16: *Event Diagram Depicting the Business Processes Interactions (used with permission) ([Weske, 2012])*

resources a company requires to fulfil its goals.

Persons are part of an organisation and they work to fulfil the business goals of the enterprise. Each person typically occupies some position, and the duties and privileges of that person come with the position, not with the person. The concept of organisational units represents the permanent groupings of persons based on their positions. Linking the organisational structure and the business processes is accomplished by activity instances called work items. Process participants are assigned to activities in a business process through *direct allocation*, *role-based allocation*, *deferred allocation*, *authorisation*, *separation of duties*, *case handling*, *history-based allocation*, or *organisational allocation*. We summarise only the first three: (i) in *direct allocation*, an individual person is allocated to all activity instances, while (ii) *role-based allocation* is the standard way of allocating work to the members of organisations based on the assumption that all participants playing a particular role are functionally equivalent; direct allocation can be simulated by role-based allocation by providing a single role with one member, and (iii) *deferred allocation* is very similar to role-based allocation, it just means that the decision about who performs an activity instance is only made at run time of the business process.

2.5.5 Workflow Management

The main concern of Workflow Management (WM) is how process structures in process models can be represented and how to control the enactment of business processes according to these models. Following the increased popularity of BPM and WM in early 90s, the Workflow Management Coalition (WfMC) was founded to bundle various workflow related activities. WfMC defines workflow management systems as follows as "Workflow is the automation of a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules" and "A workflow management system is a software system that defines, creates, and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants, and, where required, invoke the use of IT tools and applications". One important class of workflows are those that involve human interaction in part of the BP, hence parts of the business process are automated and other parts not. For a long period, modelling BP and workflows has been tackled by means of procedural specification languages, such as Business Process Modelling Notation (BPMN) [Object Management Group, 2006] for BPM, and BPEL [Andrews et al., 2003] for Web Service (WS) orchestration. Procedural approaches have critical drawbacks with respect to flexibility [Chopra and Singh, 2004], because the specification of execution requires explicit enumeration of all ordering constraints between all activities. To overcome this issue modern approaches are proposed such as the work of Pesic and van der Aalst [2006b] on modelling business processes and workflows in a declarative fashion allowing for expanded possible execution traces. Others propose a hybrid approach such as the framework of Sadiq et al. [2005] which integrates a procedural workflow specification language with unstructured parts, called "pockets of flexibility".

2.5.6 Smart Business Processes

There is an increasing number of process sectors and a great demand for the inclusion of collaborative, self-evolving, self-managed processes. As a result, business processes are required to become more intelligent especially in these three core areas: patterns, event recognition, and modelling decision making. Sinur et al. [2013] identified the necessary components to make a process intelligent as shown in Figure 2-17 and listed the following required changes:

1. *Change to Rich Outcome and Goal-Directed Processes:* Intelligent processes should be able to figure out the business outcomes and to change automatically to respond to business changes. Intelligent processes should be aware of conflicting outcomes and be able to dynamically reason and change their behaviour.
2. *Change to Rich Policy and Business Rules Management:* Traditional process could be designed to deal with rule changes, such as around navigation, workload assignment, etc.. Intelligent processes should be able to manage and change and coordinate the

change at large scale, where the rule engine is distributed and affecting multiple processes.

3. *Change to Rich On-Demand Analytic:* Traditionally processes have been restricted to a simple analytic; given the scale of data and the nature of it being distributed, intelligent business processes need to use deep poly-analytic methods to support decision-making.
4. *Change to Rich Active Analytic:* This is a new area for BP, where processes need to be proactive and able to conduct real-time event recognition, intelligent processes need to move from being only event sensitive to being completely event driven.
5. *Change to Social and Collaborative Human Interactions:* The inclusion of human resources into the business processes and utilising skills and knowledge of human actors is key to intelligent processes. Human skills and observations of areas outside the scope of the intelligent process system are invaluable to the system and are essential components that complement other intelligent business process components.

Further more, Sinur et al. [2013] argue that the intelligence of business processes can be measured by considering four aspects:

1. *Raw Intelligence:* given the exponential change, business process needs to implement intelligent ways to respond and keep up with the challenges. They have to become not only reactive to changes, but also to be pro-active and learn from its execution history.
2. *Social and Collaborative:* by design, the business processes need to make use of the distributed knowledge and employ their collective wisdom by means of collaboration and social interactions to achieve their global common goals.
3. *Agility:* Business processes needs to be more agile through the use of proactive planning and changing their goals, flow of sequence, resources, rules and exception handling responses.
4. *Autonomy:* Parts of the processes needs to remain "always on" sensing the events and assessing conditions to trigger other parts of the processes and sometimes to deactivate irrelevant goals or add new goals and start parts that are able to achieve these new goals.

2.5.7 Agent-oriented BPM

Applying agent technology to business process management is called agent-oriented BPM (aoBPM) and it has taken many directions. For example Endert et al. [2007] have proposed a translation method for the mapping from the business process modelling notation (BPMN) into agents. More recently, Guo et al. [2008] have developed a distributed multiagent workflow enacting mechanism that starts from a BPEL4WS [Andrews et al., 2003] specification to Lightweight Coordination Calculus [Robertson, 2004].

Sinur et al. [2013], claim that applying agents to BPM brings the following benefits:

1. *Distributed system architecture:* In traditional BPM there is only one process engine that controls all processes, whereas using agents enables the distribution of control and heavy use of loosely-coupled system components.

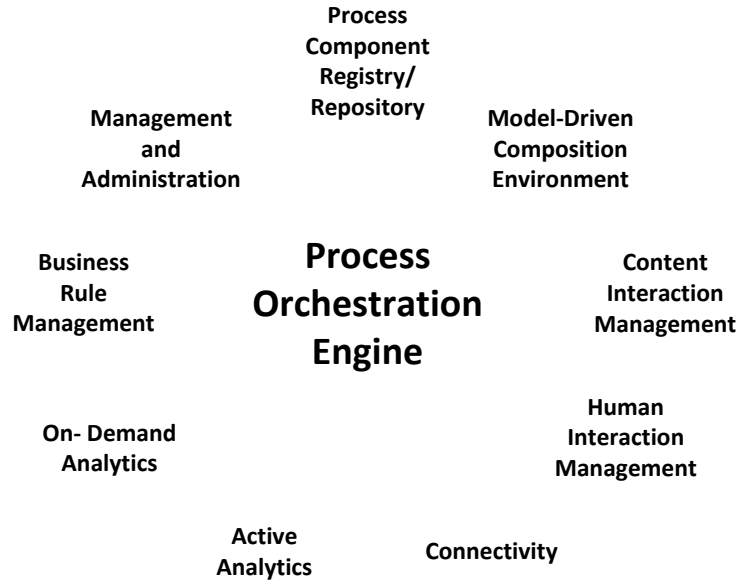


Figure 2-17: *Intelligent Business Process Management Systems Core components (used with permission) [Sinur et al., 2013]*

2. *Automation:* Agents systems are normally designed to start a work flow based on an observation of an event or a set of events that form a kind of pattern.
3. *Interaction:* Agents are able to communicate and exchange meaningful messages. Using agents allows processes to share information among them and coordinate their behaviour, while the social aspects of agents can transform the processes into collaborative organisations.
4. *Resources management:* Agents can represent humans and other kinds of resources, hence they can achieve both resource discovery and allocation.
5. *Reactivity and proactivity:* Agents are able to react to changes and in some circumstances they are able to initiate and act on plans in an anticipatory way. Employing agents transforms processes into intelligent reactive and proactive components.
6. *Interoperability between heterogeneous systems:* Agents are able to play multiple roles and interact using standard communication protocols that consist of machine-understandable messages which makes them more suitable for interoperability than APIs.

2.6 Norms and Institutions in Multiagent Systems

Multi Agent systems, as open systems, offer an environment where the system components enjoy a high degree of flexibility and freedom in deciding their own course of actions. In our view, MAS is a system with various System Participants who interact and exchange information. Human actors as one type of the available system participant types, are social intelligent creatures that use norms in all aspects of their daily social life. Therefore the use of norms is a

key part for other system participants such as autonomous agents, which are required to interact with human actors or are required to form an organisational structure with various social aspects. Norms are essential to the implementation of human and artificial agent cooperation and coordination as well as group decision making in MAS. In our work we assume that norms can be used to coordinate, organise, regulate the interactions between autonomous system components by being themselves restrictions on system participants' behaviour patterns.

Norms also serve as a language that allows for interoperability and reasoning about the normative systems models. Considering MASs as open societies, agents can not be assumed to behave in the declared or intended manner all time; self-interested agents for example might violate some of the system rules to achieve their own goals, regardless of how harmful their violation can be to the whole system or to other agents. In this scenario, where not all system participants share a common goal, agents must be assumed to be untrustworthy, where they might fail to comply, or choose not to comply with the system rules and it is essential to the system not just to define and announce sets of norms but also to monitor and observe how system participants comply or not to these norms. Furthermore the system might apply a correction or sanction action in response to any detected violation.

There is not one agreed definition of norm: some researchers consider a norm as a state to be maintained or avoided, others consider a norm as an obligation which states that something has to happen or not. In the context of our work, we adopt the later view of norms.

One of the key elements of human social intelligence is the use of norms, where the members of a group are bound to principles of right actions that serve to guide, control, or regulate proper and acceptable behaviours. Hence one way of building multiagent systems that are capable of displaying a social behaviour compatible with human intelligent behaviour and expectations, is to use norms. Boella et al. [2006] argue that integrating norms and individual intelligence in multiagent systems provides a solid model for human and artificial agent cooperation and co-ordination. Many researchers have realised the need for integrating social science theories and concepts such as norms in multiagent systems. Wooldrige, for example, presented a notion of agency that is based on flexible autonomous actions and social ability [Wooldridge, 2008].

The remaining part of this section briefly discusses the concept of norm in social theory and the definition of normative multiagent systems. Section 2.7 is a summary of related work and other approaches to implementing normative MAS. In Section 4.2.2 we present a summary of ConDec notation.

2.6.1 Norms in Social Theory

In human society, norms regulate the interactions between each individual and other individuals, groups of individuals, or the whole society. Gibbs [1965] work in the mid-1960s was among the most influential contribution to the definition of norms and their classification in social science. He noted that "a collective evaluation of behaviour in terms of what it ought to be;

a collective expectation as to what behaviour will be; and/or particular reactions to behaviour, including attempts to apply sanctions or otherwise induce a particular kind of conduct." A more recent work is the work of Therborn [2002] that presented an overview of the role of norms for social theory and analysis.

The use of norms to regulate the interactions between groups of individuals is part of what is called *social reality*. Social reality exists solely when there is a kind of joint agreement between the individuals in a society. Some elements of the social reality might be virtual, and thus norms, being part of social reality, do not physically constrain the relations between individuals. Therefore it is possible to violate them. Regulating groups and individual interactions can be done either by using prohibitions of actions that a group or an individual may want to perform, or by using obligations that order a group or an individual to perform an action at a certain moment. Obligations are normally implemented by motivating groups and individual through the use of a reward system.

Tuomela and Bonnevier-Tuomela [1995] have formally specified what a social norm is in the following form: "An individual of the *kind F* in *group G* ought to perform *task T* in *situation C*". This basic formal definition highlights the four major aspects of norms in social science: (i) individuals are different in their behaviour and may play different roles in a society, (ii) individuals are members of a group or a society, (iii) the obligation to perform or not to perform certain tasks to comply with a norm, and (iv) context-dependency of norms where they are activated only under certain conditions. The last feature is referred to in the literature as contextuality [Shimanoff, 1980]. That notion of contextuality sets also a scope to when a norm is in force or activated and when is it deactivated. This might require the system to be temporal or time-aware. We use these guidelines when specifying MSMAS norms.

2.6.2 Norms Classification:

An early classification of norms was presented by von Wright [1963] who realised the heterogenous nature of norms and proposed a classification based on what he calls the "ingredients" of norms. He lists six of them: the character ("obligation" or "permission"), the content (that which ought to be done), the condition of application, the authority, the subject (i.e. the agent to whom the prescription is given), and the occasion ("Today", "next week") [Hare, 1965]. von Wright [1963] considers the character, the content and condition of application are central, hence he classifies norms into many types among them are the following types:

Determinative Rules which define the concepts and the actions.

Technical Rules which state that some actions have to happen in order for others actions to be attained.

Prescriptions which regulate actions by making them obligatory, permitted, or prohibited. These norms should indicate who are the norm subjects, what action they should do, in what conditions and circumstances and what is the nature of their guidance (the mood).

A more recent classification is the one presented by Boella and van der Torre [2004] who classified norms into two groups, *regulative norms* that describe obligations, prohibitions and permissions, and *constitutive norms* that regulate the creation of institutional facts as well as the modification of the normative system itself. For example within the bringing about of a marriage, and the declaration: "I x by the power vested in me by the State c, I now pronounce you husband and wife", the state of being married as an institutional fact emerges from an independent ontology of "brute" physical facts through constitutive rules of the form "such and such an X counts as Y in context C" [Boella and van der Torre, 2004]. So a constitutive rule from this example is: "x counts as a presiding official in a wedding ceremony". The role of **Constitutive norms** is to construct new abstract categories, and to specify both the behaviour of a system and its evolution [Boella and van der Torre, 2004]. A system that uses norms is known as a normative system and it must specify how the system itself functions with regard to how one can change the system norms by introducing new regulative norms or new institutional categories and who is eligible to carry out such changes.

Regulative norms are used to refer to legal classification of reality where the actions are modelled like obligations and permissions attached to a classified legal categories, such as contract, money, property, marriage [Boella and der Torre, 2006]. An obligation such as sellers must ship the sold goods within 48 hours after the payments are cleared, is a regulative norm. Regulative norms are also used to express permissions, rights and powers, for example a person is allowed to sell in the marketplace if he/she represents a registered trading company. Finally regulative norms are conditional rather than categorical, they normally specify all their applicability conditions [Boella and van der Torre, 2004].

2.6.3 Dynamics of Norms

Norms motivate or restrict the system participants behaviour, but not in a static fashion. The implementation of norms can be done dynamically within a process as shown in Figure 2-18 where the norm passes through stages since its creation until it becomes abolished. As shown in Figure 2-18 Lopez and Luck [2002] suggest that a norm is issued by a legislator, then the norm is spread either by indirect or direct communication, where it can be adopted (intention to act accordingly) by the society members and it becomes part of the society. Adopted norms are normally inactive until the triggering conditions are satisfied. An activated norm does not necessarily indicate that the system participant will comply, it just sets a reference to the expected behaviour of the system participants who may choose to comply, violate or dismiss. A possible approach to regulate the system is to implement a reward mechanism that offers reward when the system participants comply with the norm or apply a sanction in case of violation. Soft/weak violations might also be ignored. Over time the system may issue new norms, modify existing ones, or abolish others.

The chosen governing body structure dictates how to implement the use of norms dynamically. One approach could be by creating a number of system administrator agents who can

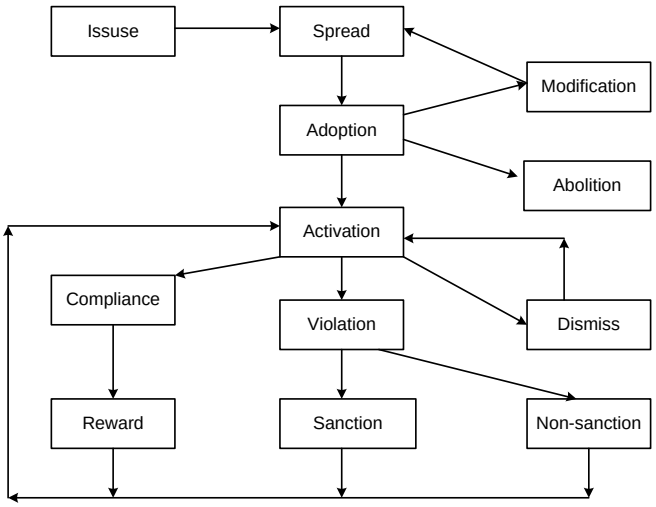


Figure 2-18: *Norm Dynamics (used with permission) [Lopez and Luck, 2002]*

observe the behaviour of other agents and be responsible for issuing obligations and enforcing them through the application of the identified reward/sanction. In this case the relationship between the system participants defines different reactions where administrator agents are just entitled to require the fulfillment of norms, and in case of violation they are empowered to to apply punishments. Lopez and Luck [2002] identify four sets of relations under this approach: (i) the first is created due to the authority of certain agents in the system, (ii) the second is created once norms become activated, (iii) the third is created once norms have been complied with, and (iv) the forth is created when there is violation of norms. Figure 2-19 shows ovals and squares that present, respectively, the norm type and the norm state, while hexagons symbolise the relationships created.

2.6.4 Normative Multi Agent Systems (nMAS)

Normative multiagent systems research is defined as the intersection of normative systems and multiagent systems. A norm is a kind of rule that has a wide applicability to a group of system participants, and norm emergence is one of the advanced properties of multiagent systems where a norm that ultimately governs a situation may not be initially apparent to the participants. Instead, norms emerge through a process of social interaction, in which agents look to others for cues indicating various possibilities of what they might expect. The main goal for using normative models is to govern the behaviour of agents through normative systems that support coordination, cooperation and decision-making [Balke et al., 2013]. Normative multiagent systems have the following properties: norms must be *explicitly represented* meaning they are formalised and codified in some form by an authority, and they *can be violated* meaning MAS must have the possibility of deviating in its actual course of action from the ideal one. If violation cannot happen then the agents are regimented and it is not an nMAS. Normative systems may require the use of violation to detect any issues with the achievement of its active

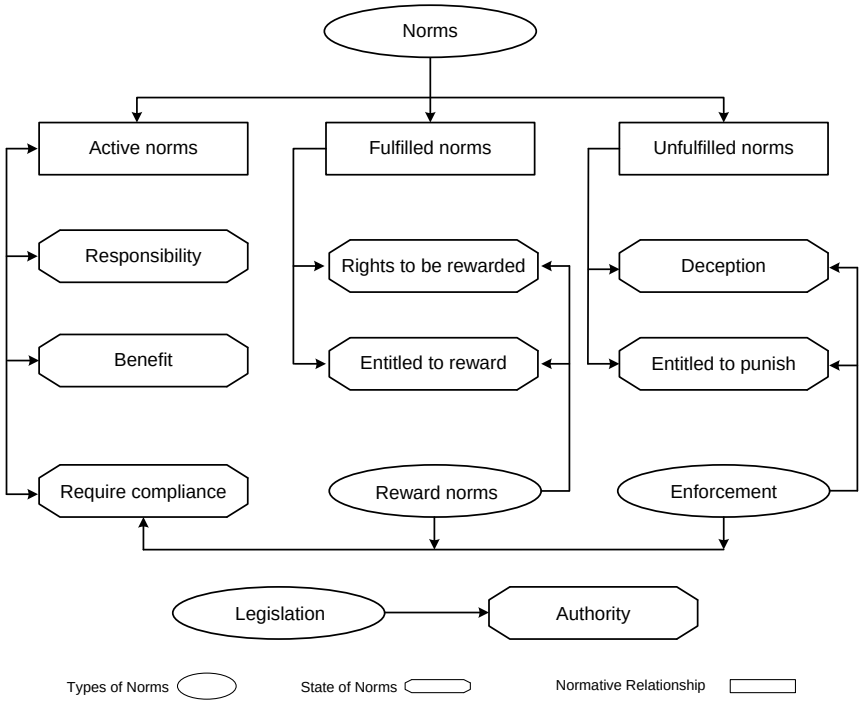


Figure 2-19: *Normative Relations (used with permission) [Lopez and Luck, 2002]*

goals. Allowing the system to recognise non-conformity with regard to its requirements enables the system to activate corrective actions to recover from the effect of violations. Guided by Hart [2012], clarification of the law and conditions under which normative systems exist.

We adopt the definition of nMAS as presented by Boella et al. [2008]: "A normative multi-agent system is a multi-agent system organised by means of mechanisms to present, communicate, distribute, detect, create, modify, and enforce norms, and mechanisms to deliberate about norms and detect norm violation and fulfilment."

According to this definition the requirements to build nMAS can be summarised as follows:

- A MAS is a system that has more than one proactive system participant.
- The system should facilitate the interaction between its system participants through specified communication protocols.
- The system participants are organised in one or more groups and these groups might be structured as well.
- The system must have a number of rules/norms that govern the behaviour of its system participants.
- The system must have means to manage the system norms, so that they can be shared, queried and changed.
- The system should be able to monitor the actions of its systems participants and reason about them according to the system norms to detect any violation and/or to check on its health in terms of norms fulfilment.

- The system could include mechanism to enforce its norms on its system participants in the way of responding to the violation by creating new norms or triggering a corrective action.

2.7 Review of Related work on Normative systems

2.7.1 Constitutive and Regulative Specifications of Commitment Protocols

Baldoni et al. [2013] propose a definition of commitment-based interaction protocols for multi-agent systems by the decoupling of the constitutive and the regulative specifications. They also define a representation of regulative specifications based on constraints among commitments using a language called *2CL* for writing regulative specifications. Commitment protocols are part of the theory of commitments in MAS that was developed over the last two decades including the work of Singh [1991] and Castelfranchi [1995]. Commitments as defined and investigated by Singh [2000] and Yolum and Singh [2001] are literals that can hold in the social state of the system. Each commitment is simply a fact that says a debtor commits to a creditor to bring about some condition.

Two of the advantages of using commitment protocols are, first, they become a shared knowledge between all agents so the semantics of a set of actions which affect the social state is common. Second, commitment protocols do not over-constrain the specification by imposing unnecessary orderings on the execution which allow agents to focus on the actual knowledge rather than on beliefs about each other. Singh and Chopra [2010] note that commitment protocols do not allow the specification of legal patterns of interaction hence they do not suit situations where the social state is constrained by conventions, laws, preferences or habits. Baldoni et al. [2013] responded by adopting the distinction put forward by Searle [2010] of two types of interaction specification: constitutive and regulative. They extend commitment-based protocols, as proposed in [Chopra and Singh, 2009] by adding an explicit regulative specification to the constitutive specification of actions, given as a set of constraints among commitments.

As shown in Figure 2-20 Baldoni et al. extend commitment-based protocols by adding a set of declarative constraints to express the regulative specifications. These constraints are expressed using *2CL*.

The 2CL Language allows for the definition of constraints based on the evolution of the social state. This approach preserves the flexibility of the model as it does not force agents to execute given paths, instead agents are free to choose their courses of actions within a range of possible actions that do not violate the rules. Figure 2-21 shows a subset of 2CL operators and their meaning.

An example model of the Contract Net Protocol [Smith, 1980] is shown in Figure 2-22 where the regulative rules are specified as follows: the initiator declares its intention to assign a task (node $n1$, $C(i,p, assignedtask(i, p))$). If this happens, the participant takes its decision

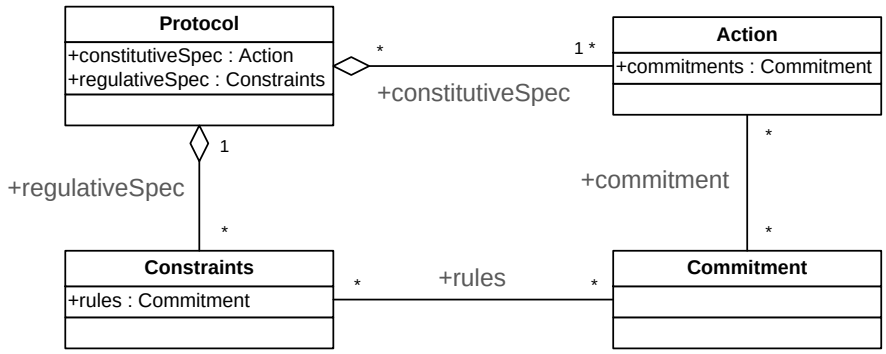


Figure 2-20: *Baldoni et al. [2013] proposal: Decoupling between Constitutive (actions) and Regulative (constraints) Specifications (used with permission)*

Response	base (pos.)	$A \bullet\!\!\rightarrow B$	If A occurs, B must hold at least once afterwards (or in the same state). It does not matter if B already held before A.
	base (neg.)	$A \bullet\!\!\not\rightarrow B$	If A holds, B cannot hold in the same state or after.
	persistence (pos.)	$A \bullet\!\!\Rightarrow B$	A persists until B becomes true.
	persistence (neg.)	$A \bullet\!\!\not\Rightarrow B$	If A occurs, it does not persist until B.

Figure 2-21: *A Subset of 2CL Operators and their Meaning [Baldoni et al., 2013] (used with permission)*

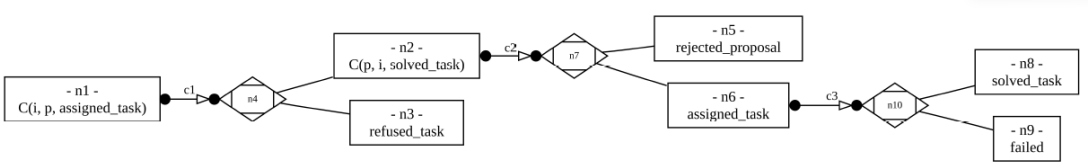


Figure 2-22: *An example 2CL model of the Regulative specification of the Contract Net Protocol (used with permission) [Baldoni et al., 2013]*

and either it refuses or states its intention to solve the task, and the protocol continues.

In this work Baldoni et al. succeed in presenting a model of commitment-based interaction protocol with an explicit presentation of both constitutive and regulative specifications and its operational semantics. The language they propose (2CL) to specify the constraints, enjoys the same advantages as its inspiration – the ConDec language – as it uses a designer-oriented graphical notation and semantics are similar to ConDec. 2CL is formalised by means of LTL. The formal representation enables the verification of interaction properties of MAS from both the global point of view and each individual agent's point of view. They argue that three basic verifications are possible; first, monitoring if the current executions comply to the specified constraints. Second, enabling agents to check the conformity of their own procedures, and third, verifying properties of the interaction protocol itself. Although Baldoni et al. advance a valid argument regarding the need to separate the constitutive commitments from regulative commitment this does not encapsulate all the necessary social aspects for the successful mod-

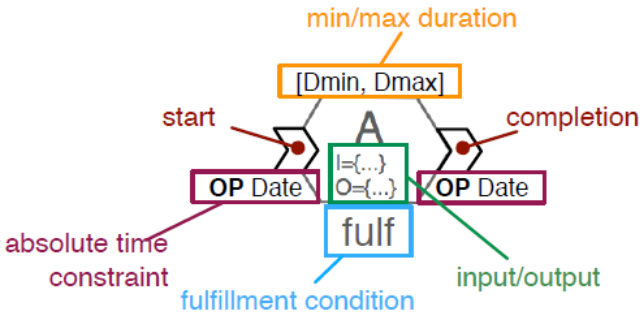


Figure 2-23: *$\mathcal{B}$-Tropos Extended Notation for Tasks and Goals (used with permission) [Bryl et al., 2008a]*

elling of MAS. Expressing communicative acts in terms of social commitments is not sufficient for the complete representation of their semantics, and system participants would not have to plan their future communicative actions in order to fulfill or violate them, unless they were formalised within an institutional framework with specification of the consequences in terms of rewards or sanctions for the fulfilment or violation of social commitments. Having this institution structure requires however to have a monitoring mechanism that is able to detect the fulfilment or violation of social commitments and enabling the governing body to enforce the system norms by applying the corrective actions either to reward or to punish the members of this institution. Lastly there is no evidence that modelling system norms in the form of commitments and contracts is suitable for modelling other non-communicative types of norms that may exist in a nMAS such as the relations between various roles that agents can play.

2.7.2 $\mathcal{B}$-Tropos

$\mathcal{B}$-Tropos is an interesting extension of the TROPOS methodology, which we summarise in Section 2.3.3. $\mathcal{B}$-Tropos adds declarative business process-oriented constructs to goals and requirements specifications. Bryl et al. [2008a] propose $\mathcal{B}$-Tropos aiming to combine requirements elicitation with declarative specification, prototyping, and analysis inside a single unified framework.

Although there is no mention of nMAS in that particular work, the authors of $\mathcal{B}$-Tropos have used declarative connections to model business processes, using connections inspired by DecSerFlow and ConDec languages, that impose a number of various constraints such as absolute time constraints, and domain-based constraint. This is a similar approach to ours with regard to the extension of ConDec semantics. They have also mapped their relations to the $\mathcal{S}$CIFF language to exploit its ability to reason about the models. The designed systems that use declarative constraints can be argued to be nMAS systems.

$\mathcal{B}$-Tropos extends the goals and tasks notation with temporal information that indicate the start and completion times. Figure 2-23 shows that each task/goal can be described in terms of its allowed duration [Dmin,Dmax], and possibly absolute temporal constraints.

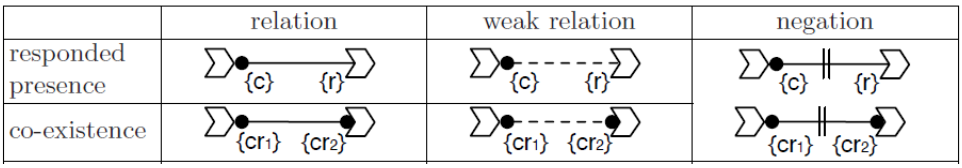


Figure 2-24: *Example Process-oriented Constraints in $\mathcal{B}$-Tropos (used with permission) [Bryl et al., 2008a]*

Furthermore, $\mathcal{B}$-Tropos extends the notation of tasks and goals relations by borrowing from ConDec three variations: relation, weak relation, and negation. The designer can then specify partial orderings between tasks with both temporal and domain-based constraints. Figure 2-24 shows and example Process-oriented Constraints in $\mathcal{B}$-Tropos. Relation and negation connections are based on ConDec template formulas with execution times constraints such as the definition of deadline, as well as the conditions that can be specified on activity start and completion times.

Bryl et al. [2008a] succeed in achieving their declared aims, and their enhancement of the TROPOS model with constraints enables the modelling of both proactive and reactive process-oriented agent behaviour. The mapping to $\mathcal{S}$CIFF allows for verifying the designed models by checking if the model satisfies a given property. Monitoring the the execution trace at run time becomes possible using similar techniques to the ones we use in MSMAS.

Although $\mathcal{B}$-Tropos shares some of our research objectives, namely to provide a requirements-driven framework for business process and agent system design and they use a computational logic for the flexible specification and verification of agent interactions, being based on TROPOS means that it is suffers some of the limitations of TROPOS original models and it lacks the required organisational rules for modelling the social aspects of agents. We question also $\mathcal{B}$-Tropos accessability and suitability for general business users.

2.7.3 Institutions

Institutions – also referred to as normative frameworks and virtual organisations elsewhere in the literature – are equivalent in multiagent systems to societal conventions and legal frameworks that govern people. They have been developed to minimise disruptive behaviour and to support the achievement of the goals for which the system was built as a result. Boella et al. [2009] define normative systems in terms of a design mechanism as:

> "A normative multiagent system is a multiagent system organised by means of mechanisms to represent, communicate, distribute, detect, create, modify, and enforce norms, and mechanisms to deliberate about norms and detect norm violation and fulfilment.".

Agent institutions have similar mechanisms to those listed in the above definition, because by definition institutions create the space of interaction opportunities between the institution

members and constrain their behaviour to allow for the accomplishment of common group goals [Fornara et al., 2013a].

Institution, as a structure, is a common organisational device that uses norms to maintain the functioning of the organisation. This particular concept is used in economics, legal theory and political science where the system uses regulation to assist human interaction at a high-level. The same principles are applied to multiagent systems in the form of the institution.

The distinction between the general concept of an organisation and that of the institution is that the institution focusses on *what can be done*, while organisation focus point is *who does it*. Hence institutions define the organisational structure that deals with norms, governance and normative events. Normative events are normally meaningful within a given social context, which is established by the institution normative model. An institutional model is a set of normative states that evolve over time subject to the occurrence of events where a normative state is a set of fluents that may be held to be true at some instant [Hopton et al., 2009]. A fluent is a term whose presence in the institutional state indicates it is true, and its absence means it is false. The main elements of an institution are: (i) a set of events that bring about changes in state, (ii) a set of fuents that characterise the state at a given instance [De Vos et al., 2011].

The institution normative fluents are normally classified as follows:

- **Permission:** fluent implies that some event may occur without violation. In case an unpermitted event occurs, a violation event is generated.
- **Normative Power:** is the normative capability for an event, without it the event may not be brought about and has no effect.
- **Obligation:** obligation fluents state that a particular event must occur before a given deadline event and is associated with a specified violation. The obligation fluent holds only when satisfied before the the corresponding deadline event occurs, otherwise it has been violated and the specified violation event is generated.

Finally, an important concept of any norm-governed system is that of *institutionalised power* [Jones and Sergot, 1996a], which means that agents that play specific institutional roles are empowered by an institution to enjoy specific relations or states of affairs with other agents.

[Dignum and Padget, 2013] express institutional constraints (norms) in the forms of: i) obligations, which are obligations to bring about a particular institutional state of affairs, once this happened the obligation is discharged ii) permissions, which define whether an action is correct or not based on some state of the institution, and iii) powers, which denote whether an action counts-as some institutional action and as a result it brings about a new institutional state. More details on institutional roles and intuitions are to be found in Chapter 4 Section 4.4.1.

2.7.4 Agent-based Institutions Frameworks

With increasing interest in developing MASs with institutions structure, a number of frameworks were proposed where each has its own advantages and disadvantages. we limit this

section to very brief mention of few of them. We have used the learning from the studying of these frameworks to inform our designing decisions of MSMAS.

OCeAN/MANET: OCeAN is a metamodel for the specification of Artificial Institutions (AIs) proposed by Fornara et al. [2008] and extended later by Tampitsikas et al. [2012] to create Multi-Agent Normative EnvironmenTs (MANET). MANET supports the development of an application-independent model of the basic institutional entities such as obligation, permission, and institutional power. MANET also can handle to some degree the norm conflicts of multiple institutions at run-time.

InstAL: InstAL [Cliffe et al., 2007b] is an action language and normative framework architecture based on Answer Set Programming [Cliffe et al., 2007a], that allows system developers to model, specify, verify and reason about institutions. InstAL meant to reduce the complexity by facilitating the task of designing an institution through translating InstAL formal description into AnsProlog for verification and reasoning tasks. InstQL [Hopton et al., 2009] is a query language that can be used with a description of an institution either in InstAL or AnsProlog to verify some of the insinuation properties.

OperA/OperettA: OperA was developed by Dignum [2003] to allow designing organisations in MAS. The OperA framework consists of three interrelated models: Organisational Model, Social Model, and Interaction Model. In OperA the objectives of an organisation are achieved through agents' actions, hence the organisation should employ the relevant agents that are capable of achieving its objectives.

OMNI: OMNI model is a recent development that is derived from OperA [Dignum, 2003] and HARMONIA [Vázquez-Salceda, 2003] frameworks. OMNI allows for the description of MAS-based organizations where agent activities are represented as agent scenes built around a collective goal. The scene actions are regulated by a set of norms. The OMNI consists of three institutional components: normative, contextual and organisational.

EIDE: Electronic Institutions Development Environment (EIDE) [Esteva et al., 2008] is a computational architecture and set of tools that are used for the designing, specification and implementation of institution-based systems. EIDE uses The EI metamodel alongside the following list of tools: ISLANDER [Esteva et al., 2002], which is a graphical specification language; SIMDEI, is a simple debugging and monitoring tool; a-BUILDER is a software that generates agent skeletons for the roles of an ISLANDER-compatible electronic institution, and AMELI a middleware for running ISLANDER-compatible electronic institutions.

2.8 Self-Management in Multiagent Systems

MASs are considered a class of distributed systems. Tanenbaum and van Steen [2002] defines distributed system as: "a collection of independent computers that appears to its users as a single coherent system". In MAS the independent system components are intelligent agents

that exist in one single system to share information, resources and services, to coordinate their actions for the achievement of a common goal, and to improve reliability and resilience of the system. The number of challenges in building distributed systems are rising because more systems are now operating in highly open, dynamic and unpredictable environments. Another factor that adds more challenges, is the increasing number of system structures that involves humans and organisations as part of the structure and operations. This class of systems faces the challenge of defining the system boundaries, incorporating both internal organisational rules and external regulations, and understanding the effect of technical components on the social one and vice-versa [Bryl, 2009]. Add to that, recently, with expectations of more frequent changes in the context/goals/requirements during run-time, the need for self-managing systems is becoming evident [Salehie and Tahvildari, 2009a]. These observations raise questions such as: how to handle heterogeneity, how to achieve openness, how to allow for scalability and how to handle failure?

Since the publication of IBM's perspective on the state of Information Technology Horn [2001] more focus was directed on solving complexity by means of developing self-managing[9] systems. Oreizy et al. [1999] define adaptive systems as: "Self-adaptive software modifies its own behaviour in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation.". the term of self-adaptive is used with the term of self-managing interchangeably, and it is difficult to draw a distinction between them [Salehie and Tahvildari, 2009b], because both share the same set of objectives and what is known as self-* properties. Sterritt et al. [2010] list the following four objectives of self-managing systems:

- **Self-configuration:** the system must have the ability to readjust itself automatically, in cases such as supporting a change in circumstances or assisting another system to meet its objectives. By definition MASs are self-configurable, as the autonomous proactive agents change their behaviours to respond to changes in their environment, and may communicate and negotiate to form a group that will collectively achieve a common goal.
- **Self-healing:** When the system encounters a fault, it must effectively recover from it. The system should be able to identify the fault, and whenever possible fix it. This is called reactive self-healing, while the proactive self-healing is when the system monitors its execution traces in search of signs indicating that the system is heading towards problems or going to reach an undesired state.
- **Self-optimisation:** When the system measures its current performance against an optimum defined performance and reacts according to a policy to improve its performance.
- **Self-protection:** To address the security risks, the system must defend itself from accidental and/or malicious attacks. The system needs to be aware of the potential threats

[9] Also known in the literature as "Autonomic" and "Self-adaptive" Systems

and has the means to identify these threats from some patterns as well as the means to handle them.

Sterritt et al. [2010] list the following system properties as essential to achieve the self-managing objectives:

- "self-aware-aware of its internal state.
- self-situated-aware of current external operating conditions.
- self-monitoring-able to detect changing circumstances.
- self-adjusting-able to adapt accordingly."

Designing a distributed system that has a degree of self-management demands considering how the system tolerates faults through their detection and recovery. In traditional distributed systems, failure handling becomes more essential as the increase in participating system components seems likely to lead to increased fault rate. The obvious solution is to add more redundancy to allow for fault tolerance and to move away from centralised control.In normative MASs, where agents may comply with or violate the system norms, violations can not be considered as failures because violations are part of the possible although undesired events. Violations of system norms could be seen, however as a partial failure. In distributed systems, a partial failure occurs when one of the components of fails unexpectedly. MASs being autonomous systems, they need to have a degree of self-management, thus a possible approach the system designer may adopt is to think of error handling as a recovery process or what Armstrong and Helen [2003] calls the "Let some other process do the error recovery" philosophy. For example, when process B monitors process A and if process A fails, process B corrects the error: in this scenario process A is a worker and process B is a supervisor. The process has to address issues such as: how to detect failures/violations? Which failures/violations can be tolerated and which must be recovered from?

Self-managing systems can be recognised from a number of properties depending on their modelling approach. For example, top-down self-managed system is often a centralised system that has a central control component that defines the policy, assesses the system behaviour and the environment and then changes or adopts a new policy that fits well with its observation, the rest of the system components then follow the new policy. On the other hand, a bottom-up self-managed system has a large number of cooperative self-managed components, and the system global behaviour is a product of the local behaviours and interactions of these components. Many approaches are proposed to incorporate adaptation mechanisms into software systems, we list and describe briefly four main approaches that are used for the building of self-managing systems. In Chapter 5 Section 4.7 we present examples on how to use MSMAS features to model self-managing systems according to each of these approaches, which we discuss in the following subsections.

2.8.1 Dynamic Components

In this approach self-managing system is designed to integrate/separate some components on-demand and the integration/separation process is performed at run-time. The system either has a repository of components it can access or is equipped with the ability to generate code to create a new component/instance. Some implementations of this approach are employing caching techniques or a directory service to reduce the time needed to search for the best-fitted component. An example of this approach is the framework proposed by Yeom and Park [2012] where they use modular agents organised as a federation of agents. Their agents are capable of re-organising and migrating themselves. The framework is a good example of dynamic components based approach and suitable for developing adaptive, highly distributed, and mobile agent-based network applications.

2.8.2 Dynamic Control

This approach depends on the use of internal/external access control mechanisms to achieve the self-management. An example of such approach is Rainbow framework which was proposed by Garlan et al. [2004] who believed that internal control mechanisms for self-adaptation are often highly specific to the application and bounds tightly to the code. The proposed framework adopts, instead, an architecture-based approach to provide a reusable infrastructure where external modules concerned of the local problem detection and resolutions can be analysed, modified, extended, and reused across different systems. Recently, Schmeck et al. [2010] describe a system with either internal or external control mechanisms to control the behaviour of the system by setting specific attributes of the system and its environment. The control mechanism can then be either a central entity, distributed or a multi-level structured.

2.8.3 Model-driven

In this approach the system is described fully/partially by a number of models, and with the help of a monitoring mechanism, the system can change between the different models and in some instances regenerate the code based on the current adopted model. An example of the approach is the work of Vogel et al. [2009] who proposed a model-driven approach for developing self-adaptive systems with self-monitoring architectures. This approach depends on a run-time system and a number of models for different self-management activities. The models are selected according to examining the collected information about the system. Another example is the work of Nakagawa et al. [2008] who utilises in their approach a requirements model to build the system architecture. The generation of system architectures is done after using descriptions of goal-oriented requirements analysis.

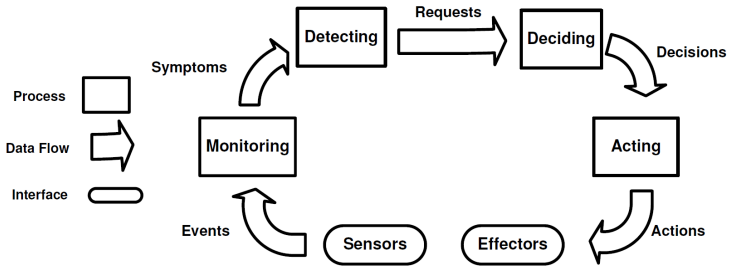


Figure 2-25: *MAPE-K Loop as proposed by (used with permission) Kephart and Chess [2003]*

2.8.4 Feedback Loops

One can argue that multiagent systems are, by definition, adaptive systems with a degree of self-management. Properties such as the loose coupling, context awareness, and persistence enable them to respond to changes as they happen and to adapt their behaviour. Goal-based loosely coupled agents provide the needed flexibility for self-adaptivity and reuse [Weyns and Georgeff, 2010]. Moreover, mobile agents based systems that have agents capable of migrating from host to another, or even duplicating themselves and sending copies to other hosts make it feasible to design a dynamic distributed self-managed systems. Franklin et al. [2006] claim that almost any autonomous system has some element of feedback control, thus we explain in a bit more detail the feedback loop mechanism.

Several solutions are proposed to incorporate adaptation mechanisms into software systems such as the use of feedback loops. In this way, an open systems implemented as an open-loop system is converted to a closed-loop system using feedback. Other solutions are feed-forward mechanisms, and workload monitoring. Feedback loops, however, is seen as having a more holistic view of what happens inside the application and its environment [Silva Souza, 2012].

Self-managing systems embody a closed-loop mechanism, which consists of a number of processes as shown in Figure 2-25. The MAPE-K loop in the context of autonomic computing includes the Monitoring, Analyzing, Planning and Executing functions, with the addition of a shared Knowledge-base [Kephart and Chess, 2003]. We now summarize these processes:

- **The Monitoring Process:** is responsible for data gathering. The data can be the system logs, execution traces, or pure events that might need to be converted to behavioural patterns and symptoms. To realise this process a monitoring process based on expectations can be employed to allow for event correlation.
- **The Detecting Process:** is responsible for analyzing the reported partial failures that are provided by the monitoring process.
- **The Deciding Process:** is responsible for determining what are the needed actions in response to the current state.
- **The Acting Process:** is responsible for applying the actions determined by the deciding process.

Roy [2007a] suggests a simple pattern that can be applied to the monitoring of distributed

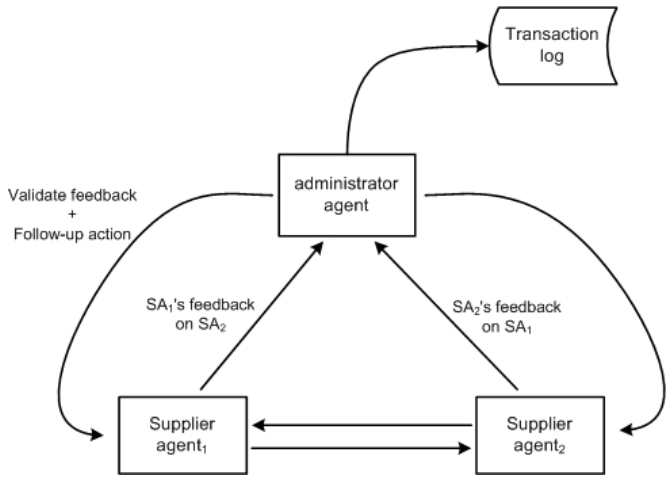


Figure 2-26: *Feedback loop example [Elakehal and Padget, 2008]*

systems so that multiple interacting control loops work together, rather than conflict with one another. He argues that systems should converge (rapidly) to the desired behaviour and not be perturbed by changes in the system's environment, but that in practice it may well collapse, oscillate or become chaotic. The difficulty is that programs typically only behave well close to or possibly even only at their equilibrium point: move the system away from that point by even the slightest amount and disaster may ensue. Roy [2007a] argues that by understanding the relationship between a system and its sub-systems, it is possible to predict system behaviour and thus design a system with the desired behaviour.

Thus, a feedback loop consists of three interacting components:

1. A component that monitors the state of a (sub-)system
2. A component that calculates a corrective action
3. A component that applies the corrective action to the (sub-)system

Figure 2-26 shows an example of a simplified feedback loop implemented in a commercial application as reported by Elakehal and Padget [2008] where *monitoring* is done by an administrator agent which receives feedback from each pair of interacting agents. Each supplier agent sends a feedback report to the administrator agent about its transaction. The *calculation* and *actuation* tasks are carried out by the administrator agent.

2.8.5 Monitoring Requirements at Run-time

From software validation and verification (V&V)[10] prospective, MASs are hard to verify not only because their behaviour may vary as a response to changes in their environment at runtime, but also because the systems evolve to satisfy their evolving dynamic goals and requirements. Verification of traditional systems are normally applied at design-time where the structural and functional aspects can be checked.

[10]Validation is the process of checking that the system operates as expected and matches its business requirements while verification is the process of checking that the system conforms to its specifications

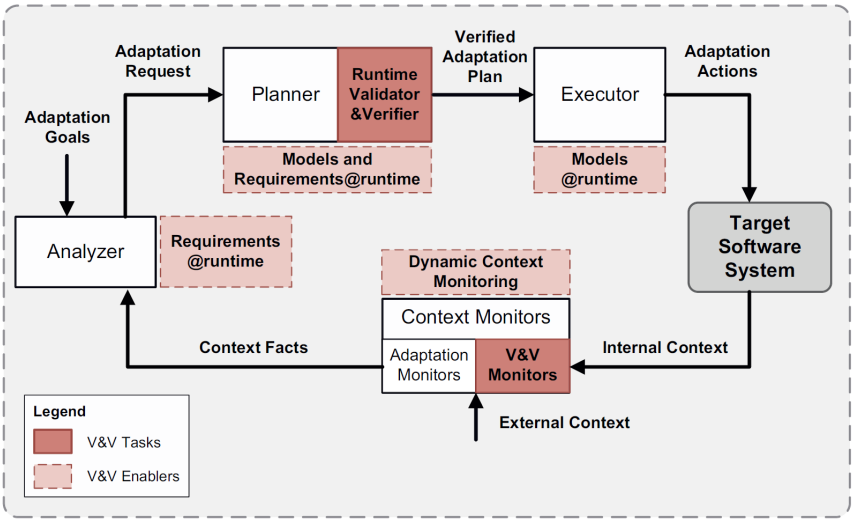


Figure 2-27: *Runtime Verification and Validation Tasks in the Engineering of Self-managing Systems (used with permission) [Tamura et al., 2013]*

Conferences and many research activities[11] took place with main focus on the verification of requirements at runtime, to identify which runtime abstractions are required and what type of modelling can be used to address the challenges raised from the volatile nature of execution of self-adaptable systems such as MAS operating in an unpredictable environment.

Figure 2-27 shows a loop of tasks as proposed by Tamura et al. [2013] to support V&V at run time. In their proposal they have a feedback loop called *the adaptation loop* with monitoring and corrective sub-systems that comprise four system components: Context Monitor, Analyzer, Planner, and Executor accompanied by two V&V Tasks and four V&V enablers. In this proposal the tasks are inline with the traditional feedback loop components, while they highlight the need for the following list of V&V enablers:

1. Enablers for the management of adaptation properties and requirements at runtime;
2. Enablers for the exploitation of models at runtime; and
3. Enablers for dynamic context monitoring.

A final observation is that to realise requirement verification at runtime the system models that capture the requirement have to be machine-processable.

2.9 Chapter Summary

In this chapter we explored a number of topics that are related to our work and needed investigation to inform our development of MSMAS. We started from the list of the basic properties of agents and MAS: being autonomous, social, reactive and proactive. Organisational structure is vital to MAS, despite the common misconceptions that agent-based systems do not require structure. Building a system with social ability reduces the system's complexity and

[11] Such as: International Conference on Runtime Verification `http://runtime-verification.org/`

can increase its efficiency. Institutions, as a mechanism, is a common organisational device that uses norms to maintain the functioning of the organisation. Institutions create the space of interaction opportunities between the institutional members and constrain their behaviour to allow for the accomplishment of the group common goals. One way of governing the behaviour of agents and to support coordination, cooperation and decision-making is through the use of norms. We discussed in detail the definition and the properties of normative multi agent systems to form a base for developing a methodology that support them.

Business Process Management involves the activities of modelling, analysis, and design. Despite the recent advances of BPM techniques with the arrival of automated business process discovery, many researchers recognise the need for business processes to become more "intelligent". The so-called Smart Business processes concept has emerged as a result. The use of agent technology can transform BPM for the better by importing BP on aspects such as implementing intelligent ways to respond not only reactively to changes, but also pro-actively, to make use of BPs *collective* knowledge by means of collaboration and social interactions, and to be more agile through the use of proactive planning. The last sections of this chapter cover Self-managing Multiagent Systems. Those adaptive systems that are characterised by one or more properties such as the system ability to readjust itself automatically, the ability to recover from faults, the ability to find and adopt an optimum performance, and the ability to address the security risks. We discussed one way to implement a system with a degree of self-management by using feedback loops that have the following processes: a monitoring process, a detection process, a decision process and an actuation process. In conclusion, we believe that online monitoring is not a feature but is essential as a means to deliver full life-cycle software validation and verification.

CHAPTER 3

REQUIREMENTS AND CONCEPTS FOR MULTI AGENT DEVELOPMENT METHODOLOGY

The first step towards achieving our main objective of proposing a new MAS development methodology is to define a set of requirements for this methodology and to define a set of concepts for the methodology to use. In this chapter we start with a list of requirements for our proposed methodology (MSMAS) followed by a presentation of a number of questions we used to identify the required list of concepts. We then present our set of chosen concepts in the form of a metamodel.

3.1 Requirements for Multi Agent Development Methodology

From the software engineering prospective, a methodology needs to include all of what is required for software engineers to enable them to do the analysis, design and implementation of their intended system. Hence, for completeness, a methodology needs to define a set of concepts, to provide an overall process to be followed and leads to the production of a collection of models and a collection of specific techniques that guide the software designer though the process [Bordini et al., 2007]. Analyzing the emerging trends from multiagent research in general, developing multiagent system methodologies in particular, and from neighbouring fields such as business process workflow modelling, and formal methods of verifying and monitoring multiagent and event based systems. Alongside studying other well known methodologies and examining their strengths and weaknesses, as well as observing the main principles of software engineering led us to compile the following list of requirements:

1. **Process Level**
   (a) It should cover the entire development life cycle.
   (b) It should have detailed description of the process steps with definitions, examples, and guidelines.

(c) It should support a wide range of application domains and the design of distributed systems with self-management ability.

(d) It is desirable to allow for iterative development and freedom of design entry points.

2. **Concepts Level**

(a) It should support the multiagent system main concepts such as autonomy, mental state, pro-activeness, reactivity.

(b) It should be limited to a number of agent oriented concepts that can specify the system.

(c) It should support some of the organisational structure concepts such as institutions, roles and system norms.

(d) It should support the concepts of modelling agents interactions; namely communication protocols and messages.

3. **Models Level**

(a) It should use visual models with notations presenting the system components and their relations.

(b) The model has to capture the concepts and their relations at an appropriate level of detail.

(c) Whenever possible allow for consistency check, refinement and reusability.

(d) Ability to present agent behaviour and agent interactions.

(e) Availability of formal representation to facilitate the verification and the validation processes.

The methodology must be detailed in describing the system and its various components and must be be complete through covering all activities and stages in all development phases. Also it must support an iterative mode of application. The availability of a supporting tool is also beneficial, although not a requirement [Bordini et al., 2007]. A tool that supports the method's various activities including design and preferably verification of design and transformation to execution code or semi-formal presentation that allows for transformation to another technology. It must support various organisational structures without limitation, and allow for methods to verify and monitor the system during design time and real-time, respectively.

3.2 MSMAS Concepts

In Section 2.4 we have reviewed a number of available metamodels for MAS and identified a number of issues, such as the lack of concepts to represent essential information and the lack of relationships that represent the system organisational structure, that made us convinced that none of the summarised metamodels is suitable for our newly proposed methodology. To make the modelling process more straightforward, we intentionally designed MSMAS to offer a carefully selected subset of MAS concepts that are essential to the description of any MAS, but especially those concepts that are more aligned and commonly used within a business context

as detailed in Section 2.5. Our aim is to keep our set of concepts balanced so it is not too detailed, to the point that modelling complexity becomes intolerable, and not too simple to the point that MSMAS models do not meet the need to specify the system clearly and allow for verification of the design, as well as to support the development and validation of execution processes at run-time.

Our view of MSMAS target systems is influenced by the the concept of socio-technical systems as defined by Giddens [1984], where such a system is bounded in time and space and shaped by social structure. In socio-technical systems there are many actors who perform actions and interact with one another in what is called the action arena [Gardner and Walker, 1994]. It is formed by interacting actors in a set of action situations. An action situation consists of roles, participants, information related to the situation, expected outcomes, costs and benefits and the actions actors perform.

Though the set of required concepts is defined through answering the following set of questions:

1. **What is the purpose of building the system and its individual components?** This question is answered through the definition of **Systems Goals** and identifying the relationships that govern these system goals. Some of the system goals could be independent and contribute directly to the fulfillment of one of the basic goals, while others contribute partially to the achievement of a bigger goal. System goals are the initial core concept that drives the modelling tasks of the system and the breakdown of each goal into sub goals is a critical process that dictates how the system components should be organised and interact.
2. **How can the system achieve its goals?** The answer of this question lies within the second core concept; the system **Business Processes** (BPs). The BPs are normally associated with one or more of the system goals and they are a set of grouped activities that form either a conceptual plan or actual execution plan. If any of these plans are executed successfully, it leads to the achievement of the associated system goal. If the business process has a number of sub business processes, then the execution of each of these sub BPs leads to the partial fulfilment of the super goal that is associated with the super BPs.
3. **Who is responsible for the execution of each business activity?** This question is answered through the definition of the third core concept of **System Participants** and their roles. The system participants can be either proactive or reactive and they come in three types (Actor - Agent - Service) within the system environment, which we see as integral part of any MAS. Few other metamodels give attention to the type of the system participants and their nature as proactive or reactive. MSMAS is similar in this respect to the work of Omicini et al. [2008] in providing a clear distinction between agents as proactive system components and reactive objects such as the explicit environmental services, which thus support the interaction between the actual agents. This approach allows for a level of abstraction that supports the design and implementation of applications that

include both human and software actors.

Each system participant typically plays one or more roles in order to be able to execute the system activities. The system participants might compete or cooperate while executing the system activities. They might be aware of each other or work independently, as required and specified by the system designer. The details of the next core concept explains more about the roles and organisational structure.

4. **What roles do the system participants play and how are these roles organised?** To address the social structure of the MAS and allow for setting a degree of control over the system components, we chose to implement explicitly the institution organisational structure, where a MSMAS system can have one or more institutions. For any system participant to take part in the system, and be able to execute any activity, the system participant needs to play the appropriate **Institutional Role** and be a member of an institution that has this role. The system participant is allowed to play more than one role and to be a member of more than one institution, however its behaviour is constrained by any limitations set on these roles. These limitations are expressed in MSMAS as system norms, which is the next core concept. Our choice of the concept of norms is motivated by the proactive nature of agents which means they can violate these kind of constraints. For example, self-interested agents may not comply with the associated system norms of the roles they play if these norms are conflicting with their own goals.
5. **What are the rules and conventions that allow the system to function correctly or to stay compliant with its design purpose?** An important objective of a MAS is the ability of more than one individual agent to interact and cooperate in pursuit of achieving common goal(s), or avoiding a conflict that could turn into an inability to achieve their own individual goals. To design a MAS that is capable of demonstrating such behaviour, we make use of institutions as organisation mechanism, so each institution member can be regulated and so they can follow specific interaction protocols, or sets of norms. Open MASs are useful because of the flexibility that allows their agents to decide on their behaviour. This however could lead to a situation where each system component is behaving in a way that drives the whole system away from its design goals, as explained in detail in Section 2.6. Using conventions and imposing regulations could limit the ability of the system components somewhat, but it can also ensure that the system collectively stays in line with its goals. In MSMAS these conventions and regulations are called **System Norms**. We have four types of System Norms which can be used by the system designers to allow/disallow certain behaviours, or even in other cases just flag a potential issue in the execution trace of the system. The system and its components can be made aware of these rules through publishing them in a public knowledge store such as the system common belief base, which is the next core concept.
6. **How does the system (and its components) store its knowledge?** For the system to operate it needs to store parts of its knowledge. These are sets of data including the

agents beliefs about themselves, about other agents and about their environment. The system **Belief Base** is simply a concept/construct that allows system components to store and share data elements. The system has a common belief base that is managed by a special system participant: the environment which is considered a blackboard where all system components can add or update beliefs or share other data in other data formats. Each system participant has also its own belief base to store privately accessible beliefs.

In summary, the MSMAS system is dependent on seven core concepts:

1. System Goals
2. Business Processes
3. System Participants
4. Communication Protocols
5. System Institutions
6. System Norms
7. Belief Bases.

In the following sections we introduce the approach of metamodelling and explain why having a metamodel is prefered for any MAS development methodology. We also present the core concepts of MSMAS, their sub-concepts, and their relations in the form of a metamodel accompanied with a detailed description of each concept group that relates to each core concept.

3.3 Introduction to Metamodelling

Any new programming paradigm needs to go through a maturity life cycle before it is fully understood and scoped, down to the best notation and methods that best suit and fulfill user needs – the needs that sparked the reasons for the paradigm to exist. The maturity life cycle starts in its first phase with big headlines where the main focus is directed towards the needs and promises; in this phase the new paradigm introduces a new method to solve unsolved problems or propose a new solution. The majority of users during this phase are usually academics and the proportion of theoretical work is normally significant. Applications designed during this phase are limited to proof-of-concepts prototypes rather than fully implemented commercial applications.

The secondary phase is usually characterised by an explosion of notations and methods. This phase deals with the tiny details of the paradigm, limits its scope and links it up to real-world problems. During this phase users encounter great troubles finding suitable methods to suit their needs. Towards the end of this phase, efforts towards the unification of modelling methods and concepts succeed in pushing the start of the third phase.

The third and final phase is the clean up phase, where only a set of the most common and practical concepts – normally unified concepts – survive and find their way into practically proven development methods. A few of these methods become mainstream.

Looking at the history of Object Oriented Programming (OOP), it could be argued that the first phase lasted for a very short period in the early 80s followed during the mid of 90s with the second phase, an explosion of object-oriented methods and notations that made it hard for its users to find a straight forward method to fully satisfy their needs. Later unification efforts on modelling languages started and ended successfully by the end of 90's with the mainstream standard: Unified Modelling Language (UML). We are enjoying a stable phase and clear view on what Object Oriented Programming is, on how to develop OOP systems, and on how to assess any system and describe it as an OOP system or not. Agent Oriented Programming (AOP)[1] is clearly following the same path but at a different pace. Despite the fact that AOP could be considered as young as OOP, AOP has not achieved the same level of maturity: it is clearly still living in its second phase. The slow progress of AOP could be due to several reasons, amongst which:

- The clarity of purpose in AOP took long debate and needed a long period to position itself away from other research branches [Wooldridge and Jennings, 1998]. AOP was, and still is, confused with AI and social sciences etc.
- Common programming problems are well understood and well served by a large OOP community, and the lack of awareness of AOP as a neat alternative to a subclass of these common problems makes AOP not popular, as yet.

The notion of the metamodel has emerged from object-oriented software engineering and is now contributing to the increasing maturity of AOP by providing a framework for the formal presentation and organisation of concepts. Metamodelling is the process of analyzing, constructing and developing of the frames, rules, constraints, visual models and theories needed for the modelling a predefined *class* of problems. The metamodel itself is used to define the languages and processes from which to form a model. According to the Meta Object Facility (MOF) standard, a metamodel is defined as a model that defines the language for expressing a model. Model-driven Architecture (MDA) defines it as a special kind of model that specifies the abstract syntax of a modelling language: it can be understood as the presentation of the class of all models expressed in that language[2]. For the purpose of our work we adopt Mellor [2004]'s definition : A metamodel is a model of a modelling language. The metamodel defines the structure, semantics and constraints for a family of models. We use UML notation as a way of presentation of our MSMAS metamodel.

A Metamodel is a widely-used representation in computer science because of its ability to display and capture the relationship between adopted concepts and as a method that allows for automatic translation of system descriptors to actual executable legacy code. Figure 3-1 shows how a metamodel is considered as a reference against which the conformity of a system model

[1]Historically Agent-Oriented Programming (AOP), was refereing to the agent-centred programming paradigm that was proposed by Shoham [1993]. In this work and as it is now generally understood, AOP refers to the general use of agents and other constructs for building software systems which include one or more software agent besides other different software components.

[2] A Proposal for an MDA Foundation Model (2005-04-01), p.2

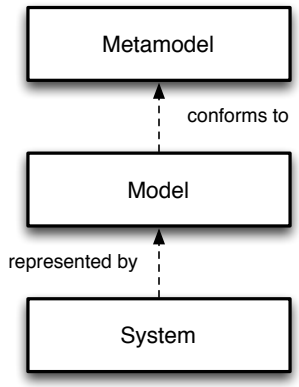


Figure 3-1: *Metamodel is the reference point to assess the conformity of a system model*

can be assessed. In other words, a system model according to a modelling method is a model that describes a system and conforms to a metamodel.

The application of a metamodel is very useful because it serves as a good presentation of the concepts and their relationships. This could support not only the transformation of system components to any target execution language, but also the mapping and transformation of a system from one modelling method to another. A well defined metamodel works as a solid foundation in all sorts of software engineering processes such as: specifying, documenting and the creation of design and development tools. Furthermore it helps in activities such as training and teaching the development method, as well as the exchange and reuse of model components, all of which are dependent on the existence of a metamodel.

Metamodels additionally help in providing a communication protocol between the modelling community in a way that helps in understanding the concepts and principles of a modelling method. Although communicating the concepts could be done through the use of standard formal vocabularies, such as mathematics, to describe the agent-based models [Hill, 2002], explaining the model in mathematics can be hard to understand for users from other disciplines, such as social sciences and business users of the method.The current approaches to agent-based modelling focus on differentiating between tool experts and domain experts, meaning that there is a recognition of the need for implementation-neutral presentations of agent-based systems. Add to that, recently the development of software engineering focused on developing methods and tools to help facilitating the development cycle. Model Driven Software Development (MDSD) [Atkinson and Kühne, 2003] as part of these developments helps in making the development of the modelling platform more accessible. The principles of the Model Driven Development (MDD) Framework of the Object Management Group (OMG)[3] are to define how a visual, model-based approach could be used to integrate a number of technologies used in software development. The core idea of MDSD is that the developed metamodel is abstract enough to allow for transformation to other platforms and yet sufficiently detailed to

[3] `http://www.omg.org/`

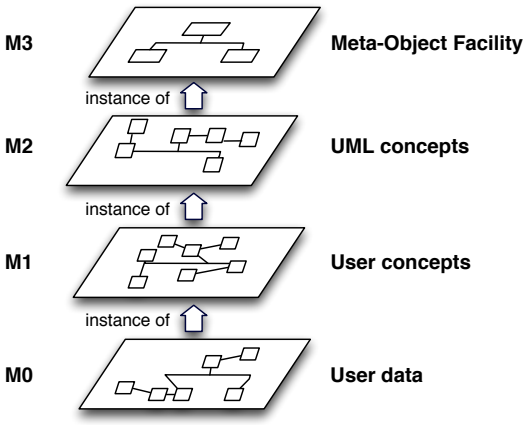


Figure 3-2: *Traditional OMG's metamoddeling infrastructure*

be able to implement it in a programming language. OMG defined a four-layer infrastructure, as shown in Figure 3-2, where the first layer is M0, which is the actual presentation of the *user data*. These data are normally built as instances of *user concepts*, which form the next layer M1, and these concepts themselves are instances of the higher layer M2, which is a UML-compliant model of the concepts, which in turn is an instance of the highest layer M3 which is the Meta-Object Facility (MOF).

In the reminder of this chapter we present the MSMAS metamodel, in which we limit our presentation to the M2 type which helps in establishing the accessibility of our method to potential users. One suitable representation to follow would be in MOF, which is not part of the requirements, hence we include it in our future work. Having a MOF model would certainly facilitate the transformation of multiagent systems, designed using MSMAS method and its tools, to another modelling platform.

The real value of the MSMAS metamodel is:

1. The definition of MSMAS concepts and their relations in a way that works as a language for analyzing and specifying a MAS and
2. Establishing a clear interface between different models at the same level of abstraction that can facilitate the process of combining or transforming parts of these models.

In the following sections we give an overview of the MSMAS metamodel, followed by a detailed presentation of each Metamodel fragment and a short comparison of each group of concepts with related metamodels reviewed in Chapter 2 Section 2.4 Page 31.

3.4 MSMAS Metamodel

Figure 3-4 shows the full metamodel of MSMAS, and Figure 3-3 is a focused view of the core concepts of MSMAS, with their immediate sub-concepts. we explain in the following subsections in further detail the underlying concepts that expand and support the core.

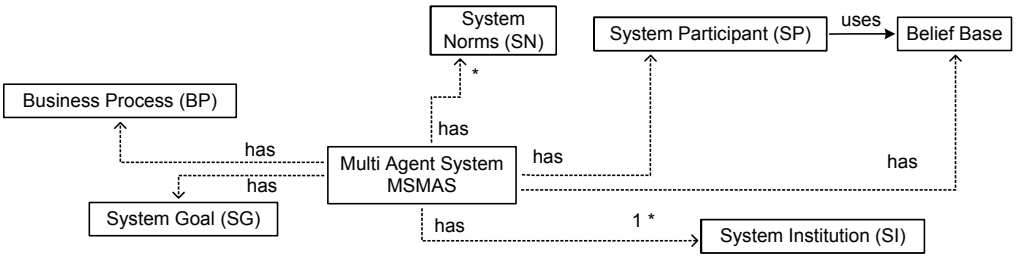


Figure 3-3: *MSMAS Metamodel: Main Components*

3.4.1 System Goals

Every MAS should have a set of goals; these are simply the motives for building such a system. The system goal within MSMAS is a state of the world that the system, or any of its participants, wish(es) to achieve. The system goals in our model are procedural, in other words the goal name is similar to a method in a traditional programming language. This is done deliberately to allow the division of – taking a top to bottom approach – the system from one unit into a group of functions. At the same time, this helps to present, in a simple way, how a particular group of sub-goals may lead to fulfilling one greater system goal.

Each system goal can be achieved by one or more system plan which is a set of business activities as defined by the business processes. Table 3.1 collects the definitions of all concepts supporting the System Goal concept. As shown in Figure 3-5, the system goal is one of three types (i) **General System Goal**: any MSMAS based system can have only one General System Goal. (ii) **Composite System Goal**: is normally a goal that needs more than one business process to be achieved, and it must have at least one sub-goal, either Composite, or Basic goal. (iii) **Basic System Goal**: is a leaf of the goal tree. it is normally a sub-goal of either the General Goal or one or more of the system Composite Goals. Basic goals cannot have sub-goals. The system designer can specify a number of System Goal Norms to express their relations such as if they are conflicting or mutually exclusive goals. For more details on system goal norms refer to Section 4.4.4.

Concept	Definition
System Goal	A desired state that the system or one or more of its participants aim individually or collectively to reach.
General System Goal	The most general reason for building the system, the achievement of all system goals leads to the achievement of the general goal.
Composite System Goal	A functional goal that can be achieved by one or more business processes, it must have one or more sub-goals either Composite or Basic.
Basic System Goal	A sub-goal of a system composite goal.

Continued on next page

Table 3.1 – continued from previous page

Concept	Definition
System Plan	An ordered list of primitive actions, that if executed successfully, lead to the achievement of its directly associated goal fully or the partial achievement of one or more of the super goals of its associated goal. A plan normally has preconditions, and the successful execution leads to change of the system's, or one of its components' state. That change of state is described as a post-condition of the plan.

Table 3.1: *MSMAS Metamodel: System Goals Concepts*

We now consider how other metamodels presented the system goal concept. JACK [Fischer et al., 2007, Papasimeon and Heinze, 2001] recognises agents as both goal-directed proactive and event-driven reactive components, hence it offers reasoning capability to figure out the needed plans, but it does not allow explicit presentation of these goals; instead goals are understood in terms of plans leading to particular outcomes that are triggered by events. Hahn et al. adopts JACK's view in their proposed unified metamodel, so instead of offering explicit system goal concepts, they utilise the concept of GoalEvents, however PIM4Agents does not present any goal-oriented concepts [Hahn et al., 2009]. In contrast FAML [Beydoun et al., 2009] proposed *SystemGoal* as a new concept. System Goals can be subdivided and related to organisations as a result to the organisation's member agents. Beydoun et al. states that including the system goals concept in the metamodel supports at design time those MAS development methodologies that exploit a goal-oriented approach, and it offers a link between organisation definition and roles. AMASON [Klügl and Davidsson, 2013] offers the concept of Mind which is expandable to include the concept of goals, in the case of adopting the BDI structure, however the current version of AMASON remains generic and does not dictate any particular structure.

3.4.2 Business Processes

Business Processes are sets of activities which are goal oriented. They describe and identify the steps needed to achieve one or more of the system goals, either fully or partially. For each **Composite System Goal** there is at least one Composite Business Process which defines one or more system plans that lead to the achievement of the associated system goal. Likewise for each Basic System Goal there is one Basic Business Process, that is a collection of a number of **Activities**. Each different sequence of activities that leads to completion of one business process constitutes a system plan. Each system plan could involve one or more system participants playing one or more institutional roles, and each institution role is considered responsible for a number of business activities either solely or collectively with other institutional roles. A collection of these activities as a system plan, to be executed by a system participant, form one system participant plan. Business activities could have a number of preconditions, and

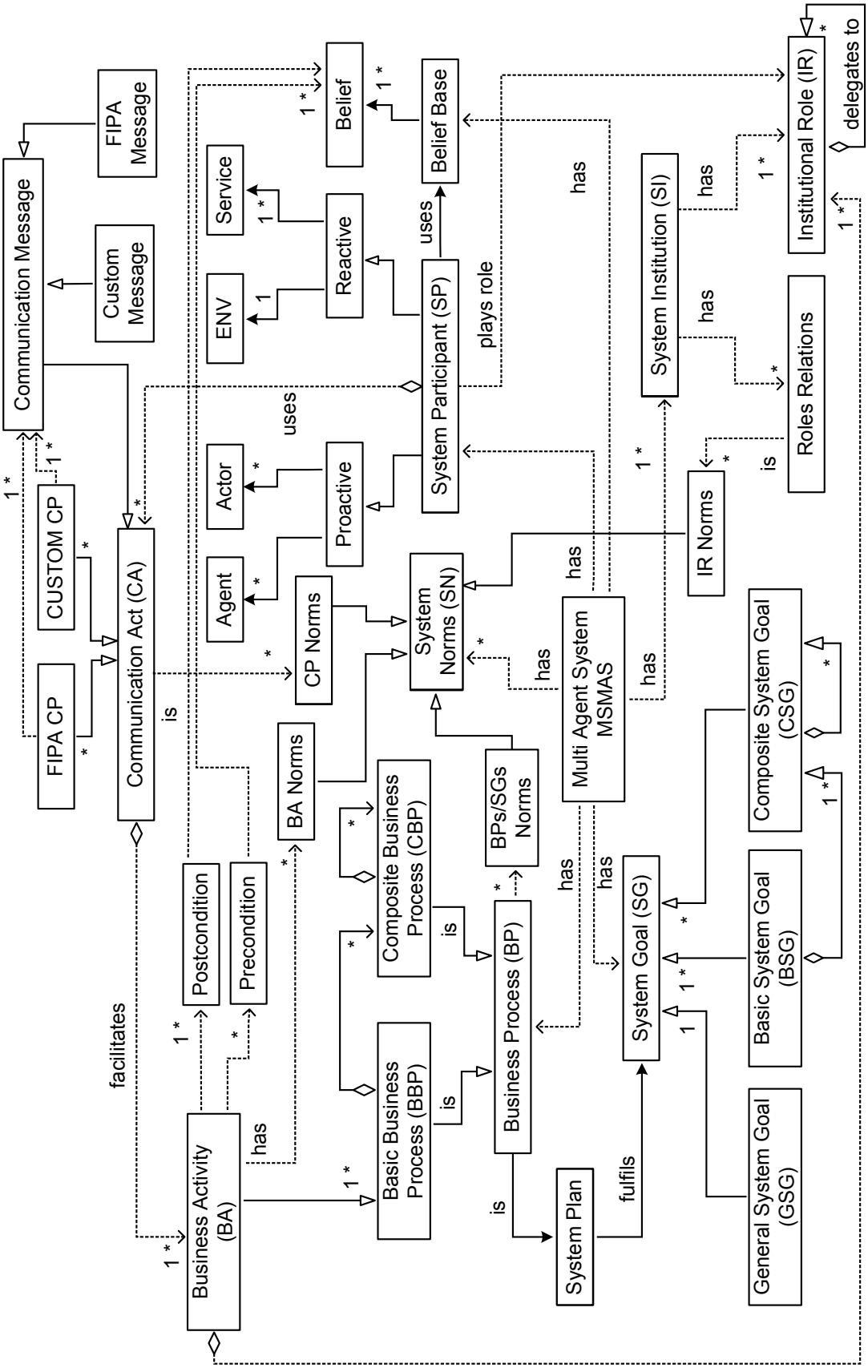


Figure 3-4: *MSMAS Metamodel*

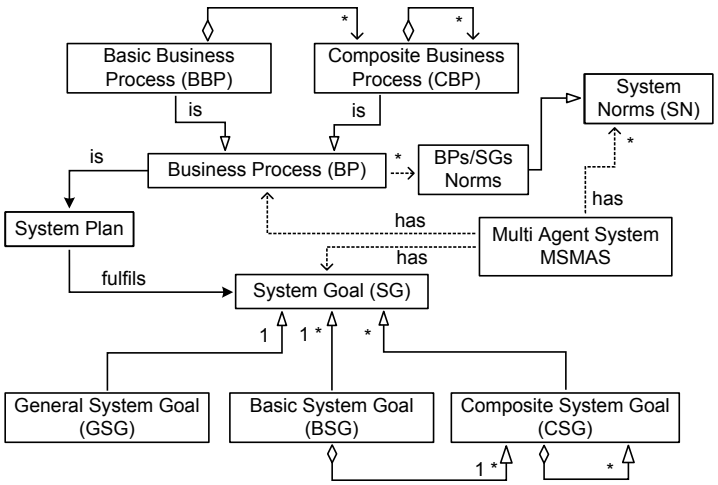


Figure 3-5: *MSMAS Metamodel: System Goals*

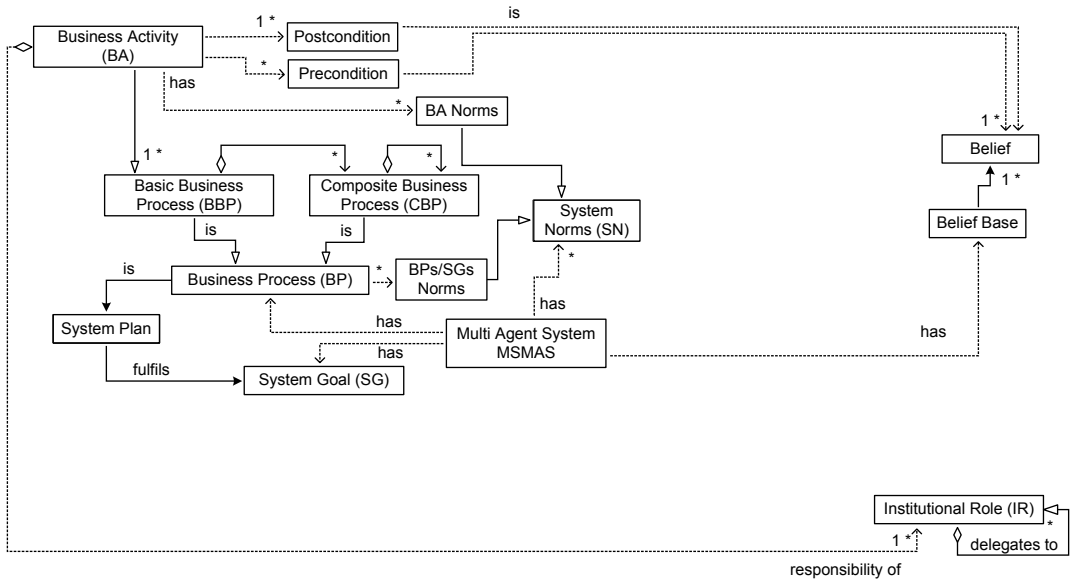


Figure 3-6: *MSMAS Metamodel: Business Processes*

each activity must have at least one postcondition. Both preconditions and postconditions are set within the system as beliefs. In MSMAS the system designer can assign system norms to both the system Business Processes and the System Business Activities (BAs), while the BA preconditions and postconditions are expressed as binary beliefs. BPs norms are considered as System Goal norms and they serve for the identification of potential conflicting goals, while BA Norms are constraints that are normally defined for the purpose of constraining the execution path of these activities. More information on System Norms is covered in Section 3.4.6. Figure 3-6 shows all the different types of Business Processes and their relations, System Plans and System Participant. Table 3.2 summarises the definitions of all concepts supporting the Business Process concept.

MSMAS does not specify what a system plan should look like, however Ghorbani et al. [2013] has suggested the following four types of plan:

1. **Atomic Plan:** a simple plan that consists of a single entity action.
2. **Sequence:** a plan that consists of a set of plans in a specific order, and needs to be executed in that order.
3. **Alternative:** a plan consisting of a set of equivalent plans from which one could be selected randomly.
4. **Loop:** a plan is repeated for as long as its associated condition holds.

Concept	Definition
Business Process	A set of activities associated with a set of organisational roles that are responsible for their execution, each activity contributes to the achievement of a defined system goal by changing the state of the system in the form of a postcondition (belief).
Composite Business Process	A collection of sub-processes or activities such that a successful execution of part or all of them leads to the achievement of a Composite System Goal. Any Composite Business Process can have one or more sub- Business process of the type Composite or Basic Business Processes.
Basic Business Process	A collection of activities such that a successful execution of part or all of them leads to the achievement of a Basic System Goal. Basic Business Process must have one or more Business Activities.
Business Activity	A primitive course of actions that involves one or more system participants playing one or more institutional roles. The BA may have a precondition and it must have one or more postcondition(s).
System Plan	An ordered list of business activities that if executed successfully, leads to the achievement of a system goal.
Precondition	A system state that has to be satisfied or become true before the business activity can be executed.
Postcondition	A system state which holds true if the business activity has been executed successfully.
BP/SG Norms	An indication of the relationship of a system goal with other system goals.
BA Norms	A constraint set on a business activity that restricts or allows particular behaviour when this activity is executed or about to be executed.
Belief	A system or a system participant's view of particular fact which is either true or false.

Table 3.2: *MSMAS Metamodel: Business Processes Concepts*

Considering other metamodels, Gaia [Wooldridge et al., 2000b] has the concept of activity that is linked with an agent role, but there is no composite concept of business process as a set of activities. In Hahn et al.'s metamodel [Hahn et al., 2009], we find the concept of activity within the behaviour viewpoint, activities are structured and can be one of three types:

Sequence, Split, and Loop. Furthermore the activity is a step within a plan and the structured activities are controlled through a FlowControl. This structure of activity concept is similar to MSMAS, but we believe that our concept structure is more powerful and expressive because of the decoupling of system norms between activities from the activity flow itself. In FAML [Beydoun et al., 2009] the concept of activity is expressed as a task in design time concepts while it is expressed as an action in the runtime. Two types of actions are defined: FacetAction and MessageAction and an action is part of a plan and each action has specifications in terms of preconditions and postconditions. This view is similar to MSMAS as it links the plans with agent definitions that are in turn associated with roles. However FAML lacks the link between communication activities and business activities: in MSMAS, we view communications activities as facilitators of the business activities that involve more than one system participant. AMASON [Klügl and Davidsson, 2013] view of activities as a type of agent-based dynamics, where the mind reasons about the needed actions, then these actions are transformed into executable actions to the body that hands then to the region.

3.4.3 System Participants

System Participants are those system components that are responsible for the execution of plans in order to achieve the system goals. As noted earlier, we expand the definition of MAS to include not only software agents and services but also human actors. The system participants that take the initiative and proactively try to achieve the system goals are called *Proactive System Participants*, whereas software services, that only respond when they receive a request are called *Reactive System Participants*. As shown in Figure 3-7, each system participant plays one or more Institutional Roles being a member of one or more System Institution The role is normally responsible for the execution of a number of System Activities that in turn, if executed successfully, lead to the achievement of their corresponding System Goals. System participants use communication acts to facilitate their interactions with other system participants, a communication act can be a single message or a sequence of messages specified by a protocol.

Each collection of activities forms a System Participant's plan and each System Participant has one or more System Plans. System participants' actions do affect the state of the system, the internal state of the system participant itself or both, and they are important for the purpose of monitoring the overall state of the system and to discover any violation of the system norms that are associated with the role this system participant is performing.

Some of these actions form the environmental dynamics, while others happen in the environment without being triggered by an agent, actor or a service. In the MSMAS metamodel this dynamic could be associated with the environment. One can also distinguish between dynamics that just affect the state of a group of the system participants and those dynamics that affect the state of whole system. We leave the management of these details to the system designer as the target implementation technology might require a different approach from one to another. System Participants have access to their Belief Base where they store and update their

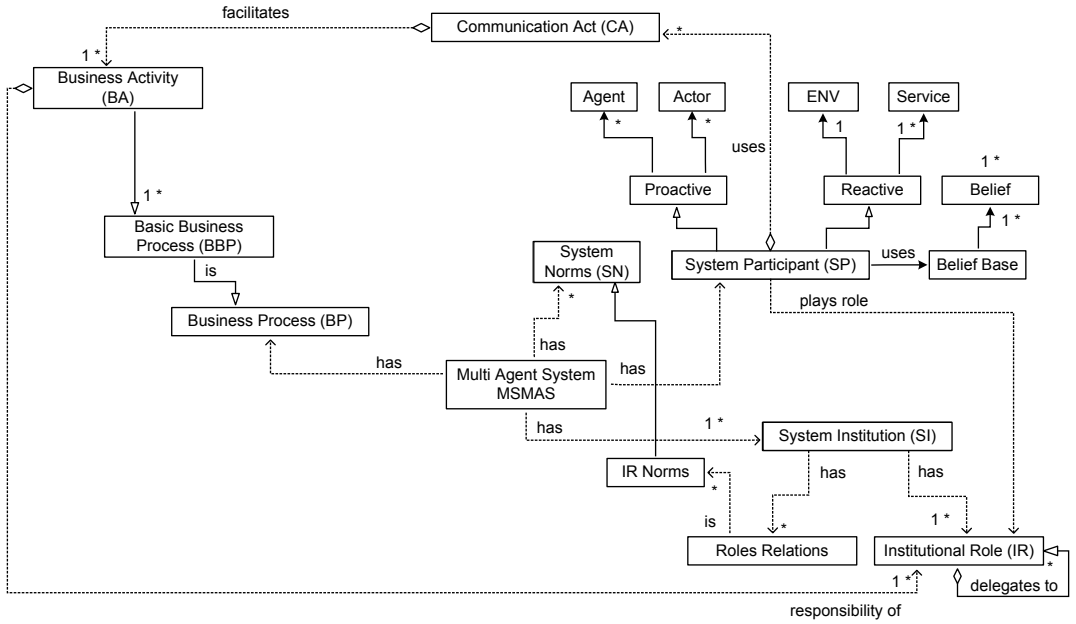


Figure 3-7: *MSMAS Metamodel: System Participants*

beliefs that are a set of facts about themselves, their environment, and/or other system participants. System Participants might also be able to access a system common belief base which is maintained by the Environment. The system participant belief base would normally contain, besides its current beliefs, its goals and committed plans. When an activity requires the system participants to enquire or share a piece of information with another system participant or with group of system participants it uses a communication protocol to facilitate the execution of this particular activity. An essential capability of the system participant is the be able to interpret and use the system communication protocols; more details on this will follow in Section 4.5.4.

Concept	Definition
Reactive System Participant	A software component that works in a stimulus-response manner: they can only respond when triggered by receiving a request.
Proactive System Participant	An autonomous software component that has knowledge of itself, environment and other components and actively uses this knowledge to reason and form/follow plans that lead to achievements of its goals
System Participants Goal	An internal goal that motivates the system participant's internal planning
Service	A reactive system participant, that has predefined set of functions other system participants can use on demand.
Agent	A software proactive system participant that actively assesses its internal state and internally plans and acts to achieve its goals
Actor	A human proactive system participant that actively assesses its internal state and internally plans and acts to achieve its goals

Continued on next page

Table 3.3 – continued from previous page

Concept	Definition
Belief Base	A store of all facts (beliefs) that a system participant holds about itself, its environment, or other system participants
Institutional Role	A specification of a behaviour pattern that the system participants should follow to carry out the function of such role
Communication Protocol	A set of rules determining the format and transmission of a sequence of data in the form of messages between two or more system participants
Belief	A fact in the form of element in the state of a system participant, environment, or both

Table 3.3: *MSMAS Metamodel: System Participants Concepts*

Considering other metamodels, the agent concept is common across all of them, however the concept of reactive system participants such as Service is not explicitly adopted. In Gaia [Wooldridge et al., 2000b], there is the main concept of AgentType which is part of an Organisation that collaborates with other AgentTypes, provides Services and plays several Roles. Hahn et al. [2009]s' metamodel adopts a minimal definition for an agent as "an entity that is capable of acting in the environment" in an autonomous manner and reacts to internal and external stimuli, and which is capable of communicating with other agents and perceiving its environment. This is consistent with our view in MSMAS, however we also allow the modelling of other system participants types such as services, environment and human actors. In FAML [Beydoun et al., 2009], the agent concept addresses the organisational aspect, considers the actions an agent takes and the resources it uses as well as the messages it exchanges. AMASON's [Klügl and Davidsson, 2013] view of the concept of an agent is as two separate entities Mind and Body, both of which are located within a region/environement: this separation is good but AMASON does not offer more details on modelling the properties of an agent and does not propose any other concepts for system participants, due to it being intended to model only agent-based simulations. We claim that MSMAS concepts are richer in respect of the types and properties of system participants.

3.4.4 System Communication Protocols

System participants need a standardised form of communication to be able to share and exchange information. In MSMAS, system participants can communicate using one or more **Communication Protocols**, which are sets of rules determining the format and transmission of a sequence of data in the form of messages between two or more system participants. MSMAS allows the use of either FIPA Interaction protocols or custom communication protocols.

The Communication Protocol is formally defined through the Communication Norms which are one of the System Norms types. The formal specification for the communication protocol

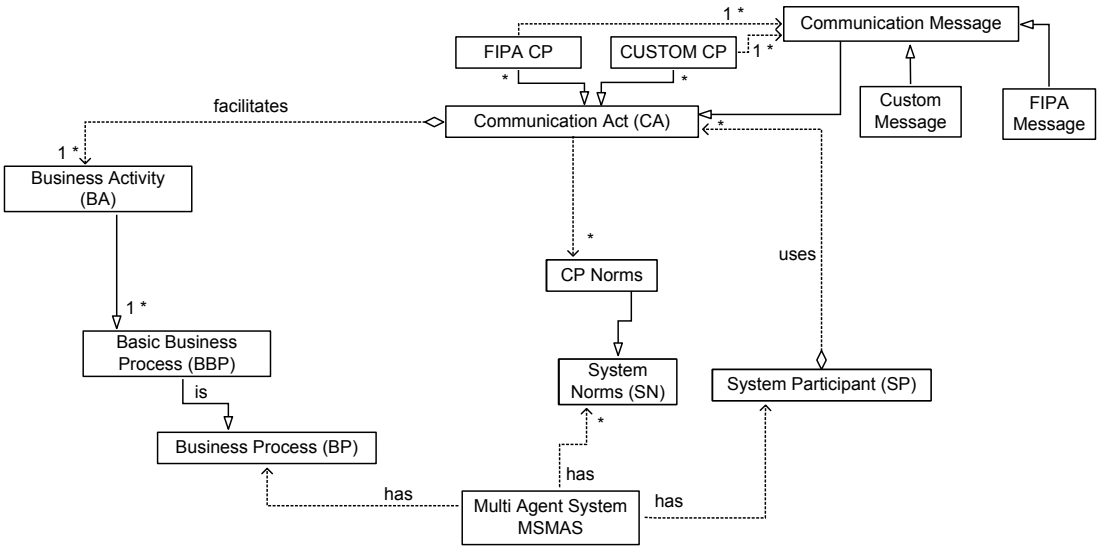


Figure 3-8: *MSMAS Metamodel: Communications Protocols*

allows for the system to discover any issues and identify undesired behaviour of any system participant during the course of an established communication dialogue with other system participants. Figure 3-8 shows the supporting concepts of the Communication Protocols and their relations, while table 3.4 shows the definitions of these concepts.

Concept	Definition
Communications Protocol	A collection of messages and rules that determine the format and exchange sequence of these messages between two or more System Participants.
Communications Message	A data container which has a standard format and can be sent from a system participant to another system participant or to a group of system participants collectively or individually.
Communication Protocol Norm	A system norm that defines a correct flow of messages between two or more system participants using a communication protocol.
Business Activity	A primitive course of actions that involves one or more system participants playing one or more institutional roles.

Table 3.4: *MSMAS Metamodel: System Communication Protocols*

Considering other metamodels, communication protocols (CP) are taken into account from different perspectives. In Gaia [Wooldridge et al., 2000b] and its extensions, the communication protocols are used by agents while playing specific roles. In each protocol, the agent may be initiator or participant, and communication protocols observe the rules set by the organisation. Hahn et al.'s metamodel [Hahn et al., 2009], consider CP in the interaction view that describes how interaction protocols are conducted between autonomous entities or organisations. The actors instantiate interaction activities and follow the protocol according the the message flow.

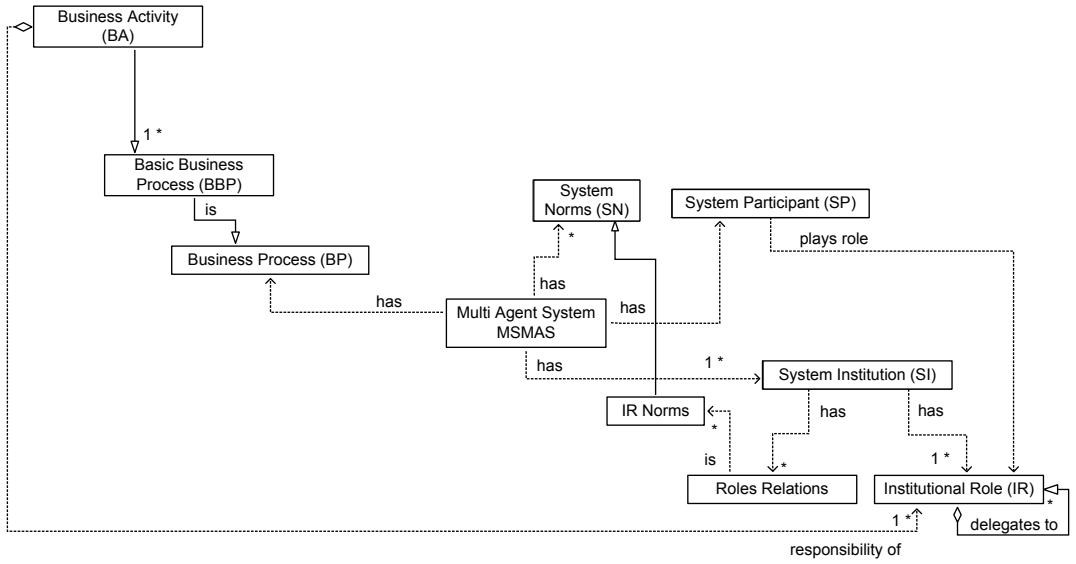


Figure 3-9: *MSMAS Metamodel: System Institutions*

Similarly, in FAML [Beydoun et al., 2009], each plan specification is a composition of a number of action specifications that could be facet action specifications or message action specifications, where the latter specifies how to send a message using a given schema. AMASON, in line with being simple and covering only generic aspects of the system, does not enforce or specify particular implementation of interactions between agents and their environment, also, lacking the organisational aspect it does not specify how agents may interact. MSMAS communication concepts are comprehensive, and more expressive to allow for coverage of wider range of applications, including simulation, and to support advanced features such as cooperation and coordination through interaction. The decoupling of system norms that specify the CP from the message itself, makes it possible for system designers to reuse the same message types in different settings by specifying different CPs that are governed by different system norms.

3.4.5 System Institutions

Modelling the social aspect of the system participants is an integral part of MSMAS. We use the Institution structure by including its concepts in the MSMAS metamodel to support the organisation structure. Each institution comprises a set of Institutional Roles. Each one of these roles can have a number of relationships with other Institutional Roles. These relations define and restrict the system participant's behaviour when they play such role. Institutional Role Norms are type of System Norm in MSMAS.

An institution itself is like a class or a template, which needs to be instantiated before being used in order to fill in the identities of the agents that it governs. In particular, and as shown in Figure 3-9 the connection between the Institutional Norms and the System Participants is established by the Institutional roles that System Participants play and their Roles relations.

Some, but not necessarily all, of the actions of an agent will relate to an institution in the sense that an agent's observable action may count as [Jones and Sergot, 1996b] institutional actions.

Concept	Definition
System Institution	An organisation structure type that contains a set of roles, set of rules and relationships that might shape the defined roles behaviour or set the line for their expected behaviour as members of this institution.
Institutional Role	A specification of a behaviour pattern that the system participants should follow to carry out the function of such role it also identifies the functional states of the system participants and their social relationships.
Roles Relation	A presentation of a System Norm / constraint that is defined between two institutional roles
Institutional Role Norm	A system Norm that describe a particular behaviour that needs to be observed by any system participant that plays that Institutional Role.

Table 3.5: *MSMAS Metamodel: System Institutions*

Considering other metamodels, each has a different stand on the organisational aspects of ab agent system. In Gaia [Wooldridge et al., 2000b], the agent is a member of an organisation with a specific structure in which the agent plays one role or more and each role has its permissions and responsibilities. In MSMAS the role is part of the organisation and the chosen structure of the organisation is the institution structure. A view similar to that of MSMAS is presented in the Hahn et al. [2009] metamodel and in FAML [Beydoun et al., 2009], where in Hahn's work the organisation view describes how single autonomous entities cooperate with others, and specifies how complex organisational structures can be modelled.

The organisational role is part of the organisation specifications and interacts accroding to rules expressed as InteractionUses.

In FAML the organisational rules govern both the organisation and the roles an agent plays as a member of such an organisation. AMASON does not specify any organisational aspect, except between the agent body and environment and how regions can be composite or presented as a grid or neighbouring regions. MSMAS institutions, in our view, offer a more flexible way to specify a number of virtual organisations and the roles shared between insitutions could be seen as a binding concept that allows for institution/institution interactions. The separation of role norms, offered in MSMAS, from other types of system norms on activities and communications permits covering of a wide range of application designs, regardless which organisation structure the system designer adopts.

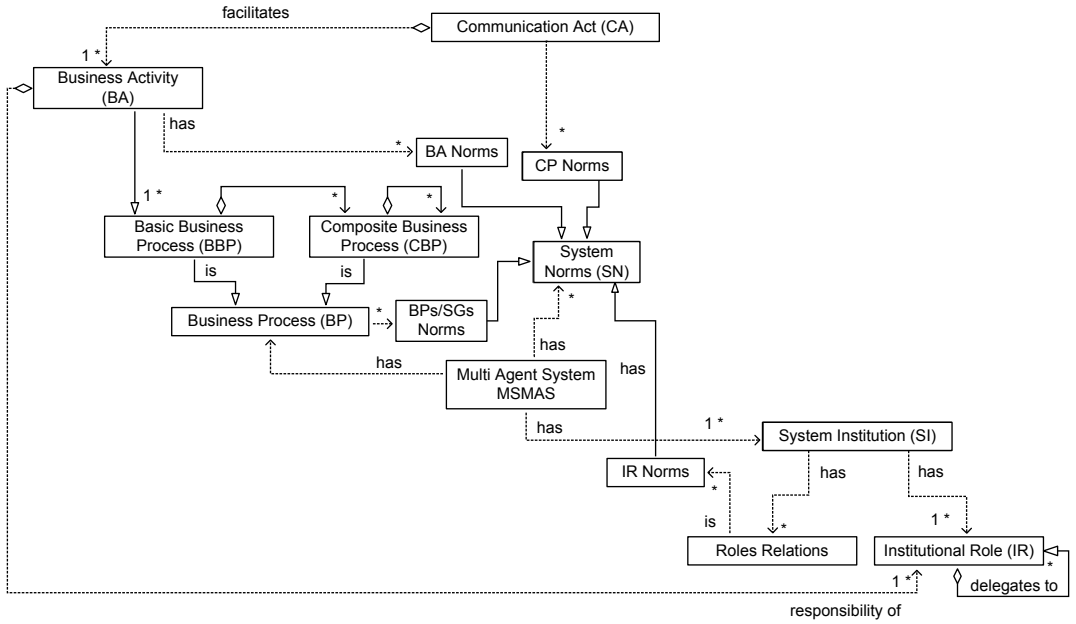


Figure 3-10: *MSMAS Metamodel: System Norms*

3.4.6 System Norms

MAS consist of interacting autonomous agents that are mainly interested in achieving their design goals, however one way to guarantee that the overall objectives of the system are observed and achieved is by regulating and organising the behaviour of the individual agents and their interactions.

MSMAS requires the use of institutions to organise the system participants in groups and it uses the institutional roles to define the behavioural patterns. Furthermore MSMAS allows for the use of System Norms, which are constraints set by the system designer on the system participants, as well as the business activities, in a way that allows a range of system participants' interactions and that restricts others.

Systems that are built in such manner are known as normative multi-agent systems: norms and organisational rules are either used by the individual agents to decide on how to behave, or are enforced by the system built-in monitoring and sanctioning mechanisms.

In MSMAS, the system designer defines the system specification by specifying the un/desired course of actions. S/he can set a number of constraints on the business activities, as well as on the system participant roles which result in defining the possible activity execution sequences and the specification of which actions can be done by a system participant playing a particular role.

To impose dynamic constraints on the activity execution, MSMAS uses ConDec^{++}[Montali, 2010], which is an extension of the graphical notation of DECLARE[4], deriving from the DecSerFlow/ConDec languages that were developed by van der Aalst and Pesic [Montali et al.,

[4] http://www.win.tue.nl/DECLARE/

2010a, Pesic and van der Aalst, 2006a, van der Aalst and Pesic, 2006]. DECLARE is a declarative language for modelling and enacting constraint-based business processes. We chose a declarative approach for MSMAS because it is well-suited to the dynamic nature of MASs, and because DECLARE offers a simple graphical notation with a powerful and flexible formal presentation, while the ConDec^{++} extension allows the expression of metric time specifications on some of these constraints.

In MSMAS models, any activity that has no constraint can be executed an arbitrary number of times in an arbitrary order as long as its preconditions are satisfied. Similarly institutional roles that have no role/role relationship can be played collectively or individually an arbitrary number of times in an arbitrary order. The constraints on system participants' roles however, are used to specify the accepted behavioural patterns when a system participant plays any of these constrained roles.

MSMAS allows for two other types of System Norms: these are the Communication Protocol Norms and System Goal Norms. The system designer can set any number of constraints on the Composite Business Processes that present the system goals, to indicate a conflict between two or more system goals; this is a useful feature as it can help in reducing the amount of wasted processing power of the system resources, when for example a number of sub goals become irrelevant or when their super goal can never be achieved due to the lack of an essential resource. In that situation the system can realise this fact and stop the cycle of attempting to achieve these (now) irrelevant system goals.

System Norms can also be used to classify agents' actions as norm-compliant or not, of which the latter may result in punishment, depending on what enforcement mechanisms are deployed. We use $CLIMB$ [Montali, 2010] which is a subset of $\mathcal{S}$CIFF to formally present the system norms where the Integrity Constraints can be used to validate the system model while action traces produced in $CLIMB$, $\mathcal{S}$CIFF, or EC format can also be used for monitoring and dynamic planning during execution as explained in Chapter 6.

Concept	Definition
System Norm	Change
Institutional Role Norm	A system Norm that describe a particular behaviour in respect of other Institutional Roles and it needs to be observed by any system participant that plays that Institutional Role.
System Goal Norm	An indication of the relationship of a system goal with other system goals.
Business Activity Norm	A constraint set on a business activity that restricts or allows particular behaviour when this activity is executed or about to be executed.
Communication Protocol Norm	A system norm that defines a correct flow of messages between two or more system participants using a communication protocol.

Table 3.6: *MSMAS Metamodel: System Norms Concepts*

Considering other metamodels, none considers a range of types of system norms. In Gaia [Wooldridge et al., 2000b], only the concept of organisation rules is equivalent to MSMAS' concept of institutional roles norms. The Hahn et al. [2009] metamodel message flow could be equivalent to MSMAS message/meassage norms, especially in that the use of ACLmessages is part of the role behaviour specification, but they do not consider explicit presentation of roles relation norms and goals norms. FAML [Beydoun et al., 2009] defines at design time an organisation that includes agent definitions which govern the roles played by agent as well as defining a role relation that express whether roles are compatible or dependent. However, it does not offer other relations between roles and there is no explicit presentation of other types of norms. AMASON [Klügl and Davidsson, 2013] leaves the concept of norms and other social aspects to future work. We believe that the expressiveness of the MSMAS metamodel of various types of system norms at many levels offers complete approach to specify organisation and permissions and restrictions on all types of system participants of agent based systems.

3.4.7 System Belief Bases

For the system to function correctly, it needs a method for sharing and exchanging information between its components. The system belief base is a collection of various belief bases that contains all facts and assumptions (beliefs) about the system participants, system norms, plans, execution states, environment etc. Each system participant has its own belief base in which it stores a number of facts/assumptions about itself, its environment, or other system participants. Each system has a common system belief base that could be used for broadcasting and sharing these facts publicly. In MSMAS, this common belief base is owned by the Environment and it can be used as a blackboard, to allow any system participant to publish a belief and make it available to others.

Concept	Definition
System Belief Base	A store of all facts (beliefs) that a system participant holds about itself, its environment, or other system participants
Belief	A fact in the form of element in the state of a system participant, environment, or both.

Table 3.7: *MSMAS Metamodel: System Belief Base*

Considering other metamodels, the Hahn et al. [2009] metamodel does not include an explicit presentation of belief, although it recognises that agents can influence the changing of the Environment, can extract meaningful information from it and can communicate indirectly via the Environment by adding to and reading information from the Environment. FAML [Beydoun et al., 2009] has an explicit presentation of beliefs as part of the agent mental state, that present its view of the environment, it lacks however a shared belief base for facilitating exchange of beliefs between system participants as proposed in MSMAS. AMASON [Klügl and Davidsson, 2013] highlights the possibility of expressing beliefs if the chosen structure is BDI,

so that the internal state of the mind would then contain current beliefs, goals, and committed plans. MSMAS concepts gather benefits of both from specifying a belief base where the system participant's beliefs are stored as part of its internal mental state, and as a common means for the exchange of beliefs, in case of the Environment belief base.

3.5 Discussion

In this chapter we have presented the MSMAS metamodel, which is a collection of concepts, that we believe are required for the development of multi agent systems. We have also compared each group of MSMAS concepts with other metamodels [§2.4], the comparison confirmed that MSMAS metamodel is more comprehensive. MSMAS covers more software engineering steps and processes and agent concepts than other methodologies [§7.4] which makes its metamodel more suitable for real-world applications and for simulation purposes. While the majority of other available metamodelling frameworks lack the social aspect and organisational structure of the system, MSMAS addresses both, allowing the system designer to extend the current concepts in fine detail to support various use cases while other interest groups could limit system designs to a high level.

Conversely, other metamodels that include organisational concepts, lack the presentation of time and spatial environment facts. Hence they do not model different classes of system participants such as agents, actors and services, instead they focus on *only* the concept of system roles and the social norms governing these roles. The MSMAS metamodel allows these system objects to be both reactive and proactive, enabling the modelling of the system objects and defining of their attributes such as which roles they play. Furthermore MSMAS separates the environment and consider it as one of the reactive system participants, allowing for defining a range of attributes that are external to other system participants.

Finally, we believe that the MSMAS metamodel is suitable for Multi-Agent Based Simulation (MABS) metamodelling. It provides clear separation between the system resources, the system participants, and their roles. MSMAS allows system designers to model the Institutional Roles that describe the behaviour patterns of the system participants, the system designer can then specify each individual object as a system resource that plays a number of these roles. MSMAS also includes the modelling of the system environment which can be limited to cover only a limited scope of the simulation. MSMAS concepts can be mapped to other metamodels that support MABS, for example mapping of AMASON's [Klügl and Davidsson, 2013] concepts can be achieved as the Body is equivalent to a MSMAS System Participant, the Mind is equivalent to the Institutional Role and Region is combination of a Belief Base, System Norms and Business Processes.

3.6 Chapter Summary

In this chapter, we identified and presented a list of requirements for developing a multiagent system development methodology. Our requirements are motivated and grounded on the need to develop a methodology that responds and resolves the issues found in the current metrologies. We have also presented a combination of business process concepts and agent concepts. Our selection of concepts is grounded on six core questions that we believe, if answered, provide a comprehensive description of any MAS, including: what are the goals of the system? how can they be achieved? who is responsible for the achievement? how are the responsible system participants organised? what constraints or system norms govern the system participants' behaviours and the business activities' execution? and how is various information stored?

We also presented our selected concepts in the form of a metamodel for MAS. The metamodel is expressed as a UML diagram that shows the concepts and their relations and follows OMG's metamodelling structure, as a first step towards a fullly OMG compliant model. Comparing the MSMAS metamodel concepts to their counterparts in other work, shows that our metamodel covers other metamodels' concepts and hence MAS systems that are designed based on the MSMAS metamodel can be mapped to other models and implemented in various technologies.

CHAPTER 4

MODELLING SELF-MANAGING MULTI AGENT SYSTEMS

4.1 Introduction

Designing and developing high quality software products requires the use of methods, general rules, techniques and a systematic work flow, which all combined are known as a Software Development Methodology (SDM). Any SDM aims to enable system designers during the early stage of development, to provide a description of the various concepts and elements that form the system, through well-defined steps. A good SDM also allows for the sharing and reusing of knowledge in a straightforward manner, and helps in improving the quality of the software product.

Modern software systems are typically distributed and they display a high degree of interaction with external software components and services over a network. But only a few of these are able to discover dynamically the required services and even fewer can exhibit degrees of self management. A self-managed system is one that can be seen as a collection of non-stop processes that are able to monitor themselves, discover any issues within their traces of events and able to apply corrective actions to resolve issues discovered.

Over the last few years a number of SDMs have been created to meet the need of software development engineers and to help in advancing each software paradigm. No different to those, agent technology has gained, since the 1980s a growing amount of interest from the research and business communities [Jennings and Sycara, 1998]. This interest has led to the development of a number of agent programming languages. The last decade has seen the proposal of many agent-oriented development methodologies that support the design and development of agent based systems. These methodologies aim to provide methods, models, techniques, and sometimes tools so that the development of agent-based systems can be formally carried out in a systematic manner.

However, the Multiagent Systems (MAS) paradigm, being a class of distributed complex systems, is more challenging, to design and develop. This is naturally due to MAS being a

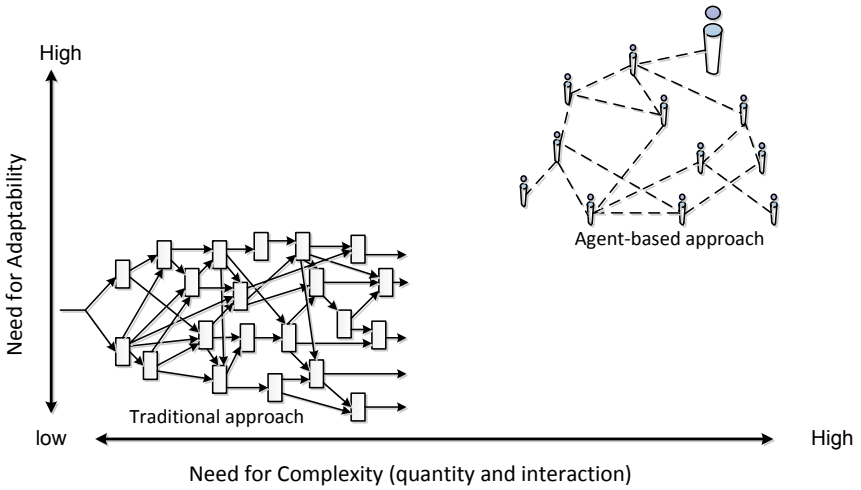


Figure 4-1: *Considering Adaptability and Complexity to Use Agents or Not [Sinur et al., 2013]*

collection of autonomous software components (agents) that should be flexible and equally communicable in a way that allows them to achieve their objectives. MAS components are normally characterised by being **autonomous**; where the agents are free to decide what set of actions to take at what times [Wooldridge and Jennings, 1995a], **flexible** behaviour that ranges between being proactive at times and reactive at others, in a way that suits the environment they operate in, and towards the achievement of their goals [Wooldridge and Jennings, 1995a]. Finally agents are capable of **dynamically communicating** between themselves, and in some cases they are expected to form new interaction patterns that constantly change [Ricci et al., 2001].

To use agents or not?

Before we explore MSMAS and our case study, we want to address one of the most common questions: when should we use agents to build the system? Sinur et al. [2013] have suggested an accessible method for the uptake of agent oriented approach, which we will follow. As shown in Figure 4-1 there are two dimensions for this consideration: the first one is the level of adaptability and agility required for the application. If the requirements of the application are all clear and are not going to change much over time and all business processes are predictable then your application is low on this axis and the use of agent technology might not be the best choice. The second axis is about how complex the application is, if the application needs to be distributed across a large number of hosts, if it has large number of components and if these components need to interact in many different ways, then the complexity is high. If your application scores high on one of these dimensions then consideration of agent technology is appropriate, and if it scores high on both, then the use of agents becomes more likely.

In Section 2.3, we have presented in some detail the classification of MAS development methodologies as well as a summary of five methodologies with an assessment of their strengths and weaknesses. Studying these methodologies amongst others helped us during the development of MSMAS to make informed decisions around the choice of steps, processes and and

concepts, as well as identified a number of issues that MSMAS has to avoid.

In this chapter we present an overview of MSMAS methodology followed by a detailed presentation of its phases and each model type accompanied with one or more example visual models and their descriptions. In the implementation phase we explain how MSMAS constructs can be mapped to executable code and give examples in one of the implementation frameworks. We also present in Section 4.7 some modelling approaches to illustrate how to design, using MSMAS, systems with self-managing properties such as self-reconfiguration, and self-control. We end this chapter with guidelines of using MSMAS and a summary.

4.2 MSMAS Methodology Overview

Modelling Self-Managing Multi Agent System (MSMAS) is a comprehensive methodology that covers the full life cycle of developing MASs consisting of three phases: System Requirements Phase, System Design Phase, and Implementation Phase. The elements of each phase and the connections between them are shown in Figure 4-2. We now outline each of the phases:

1. The first phase is principally about **System Requirements** gathering: in this phase the system designer starts by thinking about and drafting the main possible use case scenarios as well as specifying, at a high-level, the system goals that would correspond to the functional requirements, as understood from the use cases. The design artifacts generated during this phase are the **System Goals Diagram** and **Use Cases Diagrams**.
2. The second phase focusses on **Initial then Detailed Design**: in this phase the system requirements are transformed into a complete system model. It starts with the initial system design stage, which includes the high level design of the required *Business Processes* to achieve the defined (during previous phase) system goals and the specification of the system organisational structure through the definition of *Institution Models*. Then a detailed design stage that includes the design of business activities and full specification of the *System Participants* as well as the *Communication Protocols*. The design artifacts generated during this phase are *Specific Business Process Models*, *Institution Models*, *Basic Business Process Models*, *System Participants Models*, and *Communication Protocols Descriptors*.
3. Finally, the third phase concerns model verification and the implementation and execution of the designed system. In this phase the user can export the system specification in either of the two available formats. The first is the language of the CLIMB formal framework [Montali, 2010], which supports the designer in the assessment of the produced model, checking its correctness and verifying whether it meets desired properties, also taking into account possible execution traces produced by the system. The second format is the Resource Description Framework (RDF)[1] that can fully describe MSMAS system models and is used to support the transformation or mapping of the system constructs

[1] `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`, retrieved 20 January 2014

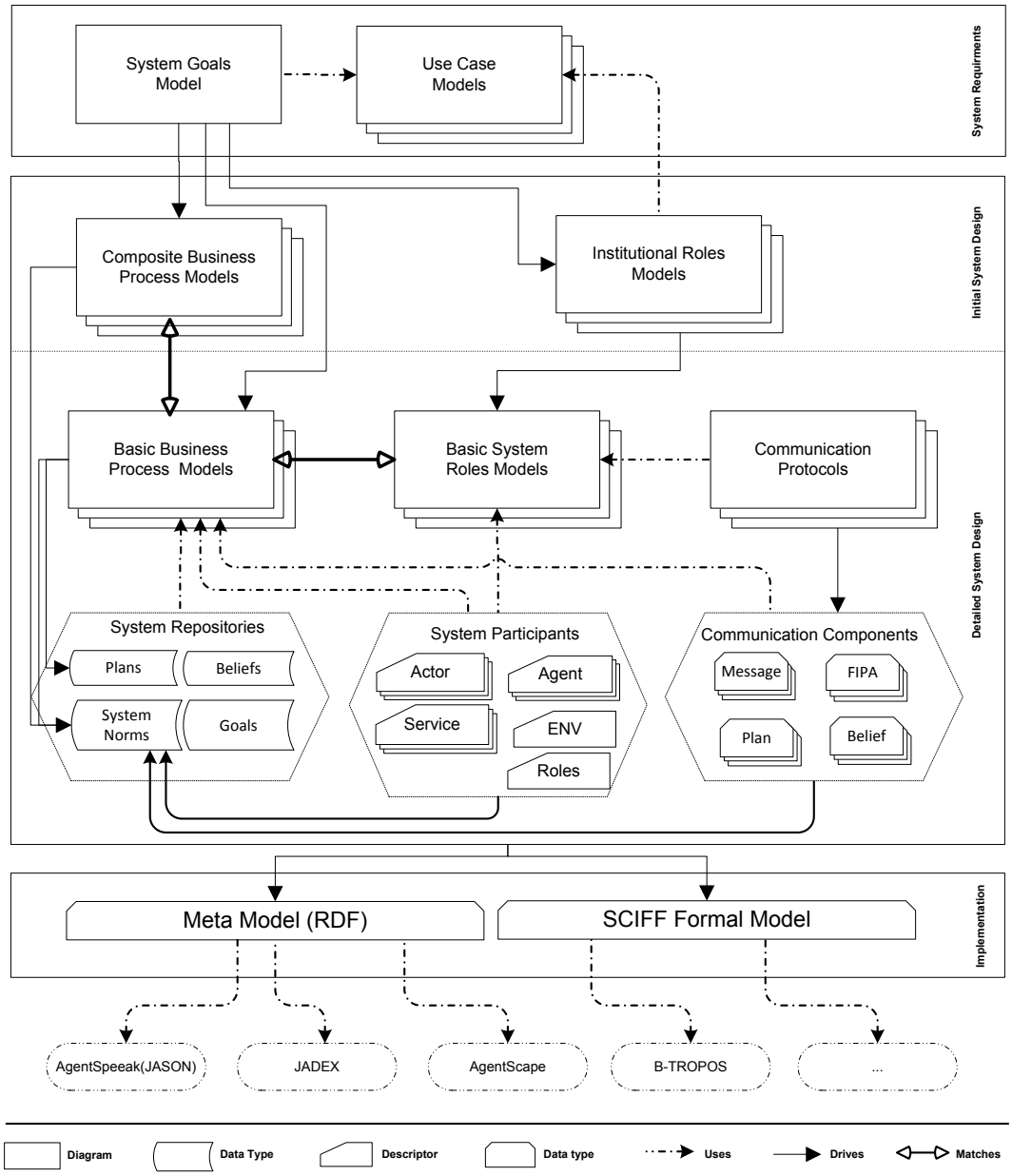


Figure 4-2: *MSMAS Overview*

into other modelling technology. It can also be used to generate legacy code in any of several possible execution languages/middlewares such as Jason, and AgentScape. We discuss and present examples of static verification of the designed models, and run-time verification of a deployed system in Chapter 6.

Although the phases of MSMAS appear to be sequential, there is a degree of flexibility in the starting points within each phase or stage. For example the system designer might start the first phase with sketching some use cases, or for more experienced users might start in building the system goals tree. In the second phase, the system designer may start detailing the Business Processes or might start from the organisational structure to define the various roles and assign

these roles to the system participants. In short MSMAS, steps do not proceed in sequence, rather, one can shift between them.

Verification of goals and other system constructs are hard if they are specified only in natural language. Regardless which domain, there are always many implicit assumptions that must be made explicit before applying formal verification [Stegers et al., 2006]. In MSMAS, we use design artifacts, to overcome issues with specification in natural languages, which are are inherently ambiguous and do not carry some certain types of information in a straightforward manner.

Before exploring MSMAS in detail, we summarise the main concepts used by MSMAS to build a multiagent system.

4.2.1 MAS and MSMAS concepts

Table 4.1 summarises the concepts that this chapter relies on. They were discussed in detail in Chapter 3.

Concept	Definition
System Goal	A desired state that the system or one or more of its participants aim individually or collectively at reaching.
General System Goal	The most general reason for building the system, the achievement of all system goals leads to the achievement of the general goal.
Composite System Goal	A functional goal that can be achieved by one or more business processes. It must have one or more sub-goals either Composite or Basic.
Basic System Goal	A functional goal that can be achieved by one or more business activities. It must not have any sub-goals and this goal type is a leaf of the goal tree. Basic goals are better when they present generic functions that can be re-used.
System Plan	An ordered list of primitive actions, that if executed successfully, lead to the achievement of a goal. A plan normally has preconditions, and the successful execution leads to change of the system state described as a post-condition.
Business Process	Is a collection of sub-processes or activities that lead to the achievement of a system goal. In MSMAS business processes are named after the goal they achieve and their sub-business processes correspond to the sub-goals of that system goal.
Composite Business Process	A collection of sub-processes or activities such that a successful execution of part or all of them leads to the achievement of a composite system goal.
Basic Business Process	A collection of activities such that a successful execution of part or all of them leads to the achievement of a basic system goal.

Continued on next page

Table 4.1 – continued from previous page

Concept	Definition
Business Activity	A primitive course of action that involves one or more system participants and may have a precondition. A successful execution leads to change of the system state, such that the post conditions of that activity hold.
Reactive System Participant	A software component that works in a stimulus-response manner: they can only respond when triggered by receiving a request.
Proactive System Participant	An autonomous software component that has knowledge of itself, environment and other components and actively uses this knowledge to reason and form plans that lead to achievements of its goals.
System Participant Goal	An internal goal that motivates the system participant's internal planning.
Service	A reactive system participant, that has predefined set of functions other system participants can use on demand.
Agent	A proactive system participant that actively assesses its internal state and plans and acts to achieve its goals. An agent normally displays a number of properties such as being situated in a dynamic environment, being autonomous, being independent and internally controlled, being responsive to the changes in its environment, being proactive in pursuing its goals and/or being sociable where it communicates and cooperates or coordinates its actions with other system participants.
Actor	A human proactive system participant that actively assesses the system state and provides some input or directional instructions to the system to achieve its goals.
Belief Base	A data store where all facts (beliefs) are held. These facts might include system participant beliefs about itself, about its environment, and/or about other system participants.
Belief	A fact in the form of an element in the state of a system participant, environment, or both.
Institution	A part of an organisation that defines the norms pertaining to a particular activity so that a number of system participants can join in becoming members and can communicate one another in a social manner. The institution might have a governor and a set of norms that restrict the behaviour of its members. System participants take part in the institution through playing one or more of the institution's defined institutional roles.
Institutional Role	A specification of a behaviour pattern that the system participants should follow in carrying out the function of such role. The role is normally specified through a number of constraints or norms, which define the permitted and forbidden actions.

Continued on next page

Table 4.1 – continued from previous page

Concept	Definition
Communication Protocol	A set of rules determining transmission of data in the form of messages between two or more system participants. An example is a file exchange protocol, where a sender system participant sends a file to a recipient system participant.
Communication Message	A communication atom which is used to exchange data between two or more system participants. The message is normally part of a defined communication protocol.

Table 4.1: *Summary of MSMAS Concepts*

4.2.2 MSMAS Norms Notation

MSMAS adopts a graphical language to represent a predefined set of constraints. We chose the declarative approach as one that enables flexibility in modelling open systems and for its suitability for formal presentation. The MSMAS methodology allows for the modelling of constraint-based systems by enabling the system designer to set a number of constraints on the business activities, the system participant roles, the system goals, and communication messages. MSMAS' choice of these four system components ensures coverage of the needed specifications with regard to system organisational and social aspects.

MSMAS uses the visual notation of ConDec^{++} to represent the declarative constraints on and between business activities. Furthermore, we extend the semantics of ConDec^{++}'s graphical notation to define a new set of constraints to be used in MSMAS models. ConDec^{++} is proposed by Montali [2010], Montali et al. [2013] as an extension of ConDec [Montali, 2010, Pesic et al., 2007, Pesic and van der Aalst, 2006a] to support better the representation of metric temporal constraints. ConDec supports a wide range of applications by including a variety of business constraints grouped into four families: *existence*, *choice*, *relation* and *negation* constraints. Figure 4-3 shows a selection of ConDec constraints with their semantics.

Figure 4-4 shows some example Metric Constraints in ConDec^{++} where ConDec constraints are marked by two non-negative time values, T_{min} and T_{max} and can be used inside parentheses (. . .) to indicate exclusion and square brackets [. . .] to indicate inclusion. In this figure ConDec^{++} combines the notation with basic relation constraints such as *response* and *precedence* to express a plethora of metric relationships between activities.

In the following sections we will explain in detail MSMAS phases, show its visual notations and give examples of its visual models. Chapter 5 also provides a number of example models to demonstrate each stage of MSMAS. In the following sections we go through MSMAS phases in detail and explore each of its models, with presentation of abstract models to show how various visual notations are used.

a — b **Responded existence** When a is executed, so must b, either before or afterwards	a — b **Responded absence** If a is executed, b can be never executed in the same case
a → b **Response** Every time a is executed, b must be eventually executed as well	a ↛ b **Negation response** When a is executed, b cannot be executed afterwards
a ⇒ b **Alternate response** Every time a is executed, b must be consequently executed, before a further occurrence of a	a ⇏ b **Negation alternate response** If a is executed twice, b cannot be executed between the two occurrences of a
a ⇒ b **Chain response** Every time a is executed, b must be executed next (i.e., as the first consequent activity)	a ⇏ b **Negation chain response** Every time a is executed, b cannot be executed next (i.e., as the first consequent activity)

Figure 4-3: *Summary of ConDec Constraints*

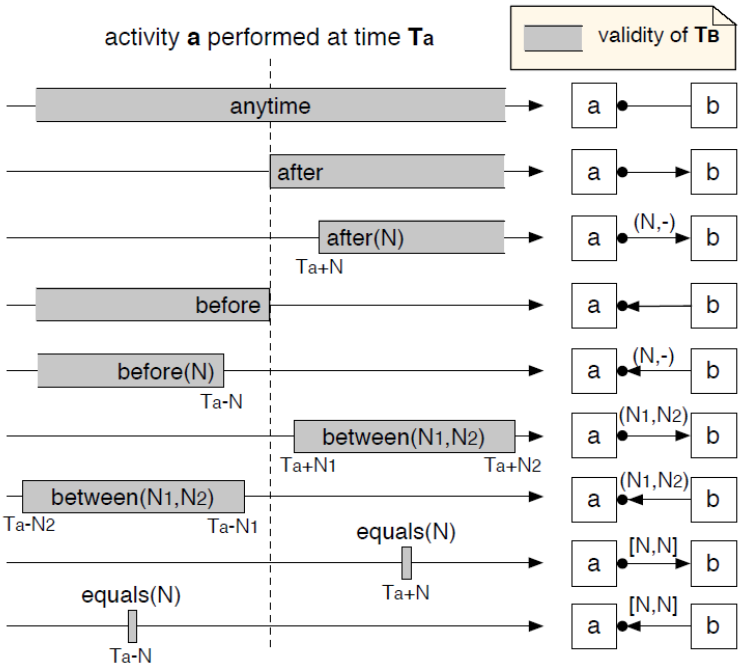


Figure 4-4: *Sample Metric Constraints in ConDec^{++} [Montali, 2010] (used with permission)*

4.3 System Requirements Phase

MSMAS method starts with collecting the requirements, followed by the analysis phase that aims at identifying and organising the system requirements and specifying its goals. There are two design artifact types in this phase, the *Use Case Models* and the *System Goals Model*. The user can choose to start with either of these models – they are mutually beneficial – although only the *System Goals Model* affects the latter design artifacts in the second phase. By the end of this phase the user has achieved:

1. a set of scenarios that have adequate coverage of the system goals.
2. a set of functionalities that are part of one or more goals, and that partially capture the

system behaviour.

3. the relationship between the system participants and the system functions
4. a set system goals and sub-goals, with their associated descriptors and structure.

4.3.1 Use Cases Models

Use Case Diagrams (UCDs) have become common design artifacts in software engineering since Jacobson et al. [1992] presented them in the context of Object-Oriented Software Engineering Method. Use Case Model properties are a complementary tool for system designers to think about the multiple scenarios that are possible in terms of system operations and potential goals. Uses cases are thought of as a clarification of some or all the system functionalities, and the models created are considered as reference documents for the later steps. The inclusion of the use case diagrams in MSMAS is strategic choice we have made to enable skill reuse and make the entry to MSMAS and MAS as new paradigms more accessible for the users of OOP, and others who are familiar with these diagrams and models. In MSMAS each use case model might present one business process and the use case diagram shows how the system can achieve one of its goals. The principle purpose of this step is to help the system designer to think through all different system functions, and be aware of any possible issues.

In MSMAS, we combine both UCDs and detailed description of the model properties in order to provide a complete description of the use case scenario, including functionality, pre-conditions, post-conditions, participants, potential goals and exceptions with the visual presentation. Among the methodologies that use UCDs, such as Agent UM (AUML) [Odell et al., 2000], Multiagent System Engineering (MaSE) [DeLoach, 2004], and PASSI [Cossentino and Potts, 2002], each lacks some properties or detailed description which means they are not able to collect a sufficient description of the requirements.

The use case diagram normally shows how different actors interact, which steps they take to carry out one system function and the possible function execution order. The user can start thinking about all use cases, then refine these into the possible use cases and exclude the impractical ones. Then s/he might extend the use case by filling in more details about its description and identify any association between use cases. Following this process helps the user to validate the requirements through exploring multiple scenarios. Although use case models are not prerequisites for other models, it is not advisable to skip these models. The system designer should create the minimum number of use case models that reflects the core functions of the system in questions to clarify those areas that might be ambiguous to other users and developers. At the same time it is not advisable to create a use case model for each system function because this can be a time consuming process and for those obvious use cases the model offers little value.

In the process of developing the use case, it is common to identify the system goals to support the use case, it is also common that the system designer re-considers how already defined system goals are organised, or re-used, or even to check if they were the appropriate

ones for the system functions and if they are in their natural order. It may be the case that there is a need to introduce a new system goal, functionality or system participant.

Each use case model must have the properties in Table 4.2 alongside the visual use case model.

Property Name	**Data Type**	**Description**
Use Case ID	INT	Auto incremented system generated ID.
Use Case Name	String	Unique Name.
Description	String	More details to describe this use case.
Goals	List	A list of possible system goals identified to support this use case.
Preconditions	List	A list preconditions to be met before the start of this use case functions, or the triggering events/-beliefs that activate one or more functions within this use case.
System Participants	List	A list of possible system participants taking part in the functions identified in this use case.
System Functions	List	A list of functions used in this use case.
Normal Flow	List	A list of a normal course of actions or the execution sequence when the system behaves as desired.
Exceptions	List	A list of possible points of failure.
Notes	String	Any additional information the system designer needs to note about this particular use case.

Table 4.2: *Use Case Model Properties*

To create the use case model, the system modeller should consider doing the following:

1. Give an indicative name to the use case.
2. Describe in detail the scenario that is covered by this use case, and refer to other use cases that might complement, proceed or follow this use case.
3. Create a list of system goals covered by this use case, and refer to other use cases that have dependent system goals on the goals of this use case. System goals should be organised hierarchically whenever possible.
4. List all preconditions for all functions in the use case
5. Identify the system participants that take part in the use case functions, consider their types and name them accordingly.
6. List all use case functions with a description of each function.
7. Describe the normal flow of execution in the use case.
8. Identify any exceptions and describe the recovery steps if any.

Figure 4-5 shows the diagrammatic notations of Use Case Model while Figure 4-6 shows an abstract use case model that describes a scenario involving four system participants (SP1, SP2, SP3 and SP3) and three functions (Function1, Function2, and Function3). The solid lines

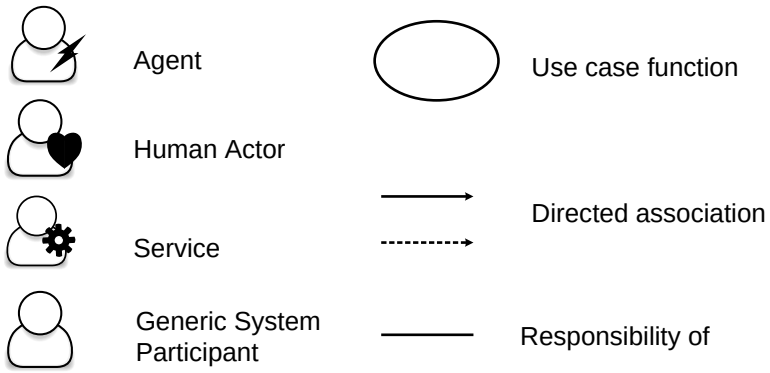


Figure 4-5: *MSMAS: Use Case Model Notations*

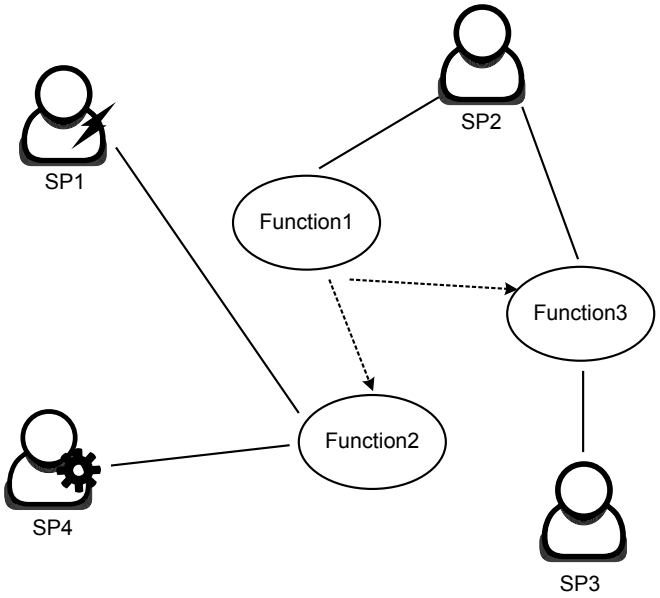


Figure 4-6: *MSMAS: An Abstract Use Case Model*

indicate which system participants responsible for which functions. For example in this particular sample model the system participant (SP2) is responsible individually for (Function1) and responsible collectively with the system participant (SP3) for (Function3). The notation used for SP1 shows that its type is an Agent, and the notation used for SP4 indicates that it is a service while SP2 and SP3's types are not decided yet. The arrows between the functions indicate the sequence of execution and the dotted arrows indicate that these are alternative execution routes.

For a concrete example based on real-world scenario please refer to case study in Chapter 5 where Figure 5-1 and Figure 5-9 show the visual use case models, while Table 5.2 and Table 5.15 show the detailed properties of these models.

4.3.2 System Goals Model

Using goals to model business processes and complex systems such as MAS has become a common approach since Bratman's belief-desire-intention model (BDI) [Bratman, 1987], and its development by Rao et al. [1995], so that many MAS modelling methodologies are inspired by and have taken BDI concepts as their core modelling concepts.

Yu and Mylopoulos [1994] proposes a modelling framework that uses goals and rules to analyze and to design Business Processes. Their work follows the means-ends reasoning technique, where they consider a goal as an attribute to a business activity, to allow any system participant to select the activity that achieves that goal. There could be, however, more than one activity that matches the goal so to solve this issue they allow sub-goals and subtasks, where a sub-goal leads to the search for an activity, while a sub-task names a particular activity without search. In MSMAS, we consider the definition of goals and sub-goals to be a core task that leads to the definition of BPs and their activities. The decomposition of goals into sub-goals is a rational task that helps the designer to break down the system into smaller pieces, to the granularity level that suits its resources. System goals are a natural construct to use in system specification and they are central to the functioning of the intelligent software agents [Padgham and Winikoff, 2002]. The use of goals and definition of their relations at the requirements engineering and system analysis phase facilitate the mapping later in both initial and detailed design phases. We define system goals as long term goals that present the final state that the system wants to achieve to satisfy its business objectives. System goals can be built as a hierarchy of sub-goals where their scope narrows down and one or more sub-goals combine to fulfil their super goal.

Every system should have a set of goals: these are simply the motivations for building the system. Using the System Goals Model, the system designer does not have to specify more than the system goals and the goal hierarchy. The goals are arranged into a tree where the leaves are the most detailed level and where every goal can be fulfilled by *only* one basic business process. System goals are the drivers for all the diagrams in the next phase.

The system goal is ultimately the state the system wishes to reach. The system goal definition should not to be confused with common agent goals: in our model the system goals are procedural, in other words the goal name is similar to a method in a traditional programming language. This is useful to divide – if we take a top to bottom approach – the system from one unit into a group of functions. At the same time, it helps to show how a particular group of actions may lead to the fulfilment of a single big system function. Figure 4-7 shows the diagrammatic notations of System Goals Model, while Figure 4-8 shows an example System Goals Model. The system goals hierarchy in the system goals model shows the sets of sub-goals that lead either individually, or collectively, to the achievement of their super goal. MSMAS allows for the presentation of "and" and "or" relationship through the use of System Goals Norms at the composite business process models level, which is discussed in detail in Section 4.4.4.

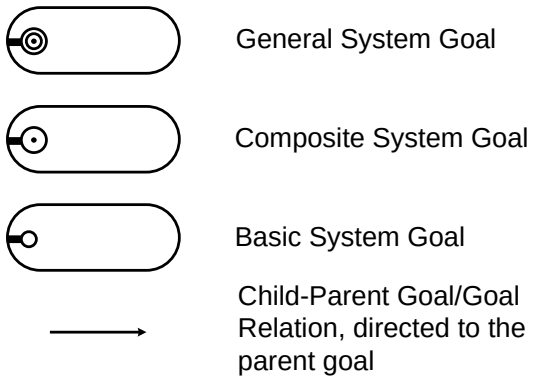


Figure 4-7: *MSMAS: System Goals Model Notations*

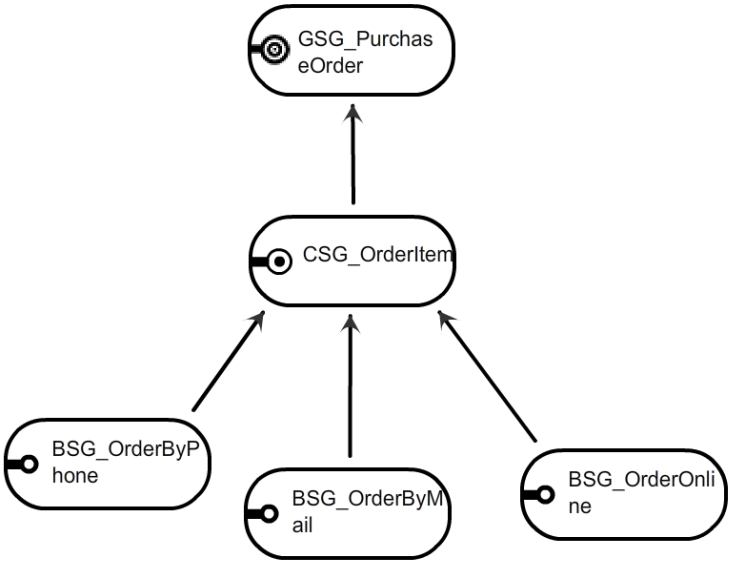


Figure 4-8: *Example System Goals Model of the Purchase Goods System Segment*

To give and example of modelling equivalent sub-goals that create a number of alternative routes for achieving a super goal, consider a sub-system for the purchase process where the general goal is the "Buy an item" and one of the sub-goals is the "order an item". The system designer might allow the user to order through one of the available options (OrderByPhone, OrderByMail or OrderOnline). Modelling this system segment can be done through building the goals tree as shown in Figure 4-8 and setting the system goals norms that define these three sub-goals (OrderByPhone, OrderByMail or OrderOnline) as disjoint goals as shown in Figure 4-10. The diagrammatic notations of Composite Business Process Models are shown in Figure 4-9. Reasoning about these System Norms is discussed in more detail in Chapter 6.

Creating the system goals model depends heavily on a system description that normally contains implicit and explicit indications of system goals and the use case models that were created in the previous step. The designer should now consider segregating these goals into three types: some goals will end up as composite goals, others as basic goals, and the rest will

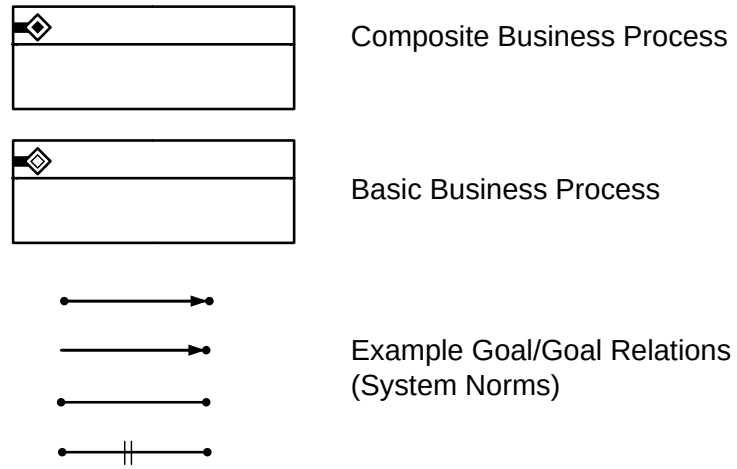


Figure 4-9: *MSMAS: Composite Business Process Model Notations*

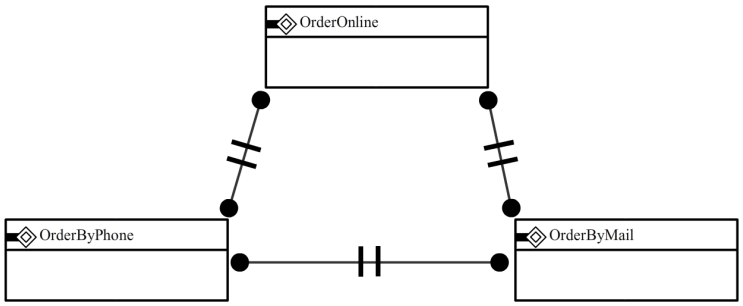


Figure 4-10: *Disjoint System Goals Norms Example*

be activities within a basic goal. If the goal can be broken down into a number of sub-goals then it is either a composite goal or a basic goal. If it is too generic and its scope is too wide, then it is definitely a composite goal, while a basic goal should present only one function.

Goals are normally organised as a tree as an AND/OR refinement-abstraction structure where the higher level goals are strategic, coarse-grained goals that involve multiple agents [Dardenne et al., 1993] whereas lower level goals are technical, fine-grained goals that involve fewer agents [Darimont and Van Lamsweerde, 1996]. The breakdown of each goal to its sub-goals is called *refinement* and it aims normally to reach all subgoals that are sufficient for the system domain and can be managed, where AND refinement refers to all sub-goals that are sufficient to satisfy the system super goal. OR-refinement, in contrast, establishes the link that relates a goal to an alternative set of refinements/sub-goals. The goal refinement process ends when every subgoal is realisable by some agents and is expressible in terms of conditions, controllable [Letier and Van Lamsweerde, 2002] and feasible in terms of implementation. Goal refinement can be done by "*how*" technique, where for each goal the designer asks the question "how can this goal be achieved?" [Van Lamsweerde, 2001].

Following the refinement process comes the categorisation of goals as basic or as an ac-

tivity. The designer asks, is this goal useful and contributing to the achievement of more than one super-goal? Can it be re-used outside the scope of its super-goal? If the answer to both of these questions is "yes" then it is a basic goal, otherwise it is better to model it as an activity.

Another source of help in creating the system goals model is the use case models, where some potential system goals are identified with their sub-goals. Not all goals identified in the use case models make their way to the system goal tree however, some of them may be redundant, being equivalent to other goals. Other goals might also be well placed as an activity within a basic business process.

As a final step the designer reviews the system goal tree and checks for any of the following issues:

1. Are there any composite goals that have only one basic sub-goal? Can that goal be removed and replaced by its sub-goal?
2. Is there any goal repeated twice or more?
3. Is there any goal that has no super-goal apart from the general goal?
4. Is there any composite goal that has no sub-goals?

Each system goals has the list of properties shown in Table 4.3. The system designer should always complete these properties in the goal descriptor to facilitate the process of exporting his/her design in one of the supported formats as discussed in Section 4.6.

Property Name	**Data Type**	**Description**
Goal ID	INT	Auto incremented system generated ID.
Goal Name	String	Unique Name.
Type	String	General, Composite, or Basic.
Description	String	More details to describe this system goal.
Super Goals	Array	List of super goals, applicable for *only* Specific and Basic goal types.
Sub Goals	Array	List of sub goals, applicable for *only* General and Specific goal types.
Fulfilled By	String	Name of the Business Process associated with this system goal.

Table 4.3: *System Goal Properties in MSMAS*

The system has only *one* System Goals Model: these goals are directly linked to the business process models and any change in them affects the business process models. Table 4.4 shows the properties of the System Goals Model in MSMAS.

Property Name	**Data Type**	**Description**
Goals Count	INT	The total number of system goals.
Verified	Boolean	Set to True once the model is verified otherwise it remains false.
Created on Date	Date	In the format ddmmyyyy.

Continued on next page

Table 4.4 – continued from previous page

Property Name	Data Type	Description
Created on Time	Time	Time formatted as hhmmss.

Table 4.4: *MSMAS System Goals Model Properties*

Completing the system goals model and the goal descriptors concludes the first phase of MSMAS (System Requirements Phase). By now the system modeller has created:

- a set of scenarios that cover the main system goals, system functions, and system participants.
- a detailed set system goals and sub-goals, with their associated descriptors and structure.

The next section covers stage one of the System Design Phase, where we will discuss the creation of the organisation structure and the refinement of system norms.

4.4 System Design Phase: Initial Design Stage

The second phase of MSMAS have two stages. The first stage focuses on giving the system an organisational structure through the specification of institutional roles and allocating them within one or more institutions to create the Institutional Roles Models artifacts. This is followed by the creation and refinement of the system norms at the institutional roles level, as well as at the goals level, by declaring role/role relations between the institutional roles and goal/goal relations between the BPs in the Composite BP models. By the end of this stage the system modeler has achieved:

1. Development of a set of institutions, where each one groups a set of system roles.
2. Definition of a set of system norms at the system roles level by setting relations that govern the system roles, and form the structure of the system organisation.
3. Preliminary assignment of system roles to system participants.
4. Refinement of system goals structure with a set of required system norms at the system goals level by defining relations between the business processes linked with system goals.

In the following sections we cover these activities in detail and show example models and descriptions based on “Virtual Stock Control and Offers Management System” case study.

4.4.1 Institutional Roles Models

Introducing organisational structure to MAS is essential to address the social aspects of system participants, as discussed in section 2.2.4 Page 17. Modelling business process applications requires the inclusion of a company’s organisational structure [Weske, 2012], and the description of the behaviour of the problem solving components. MSMAS uses the notion of institution and institutional roles to allow the BP modeler to implement the organisational structure that

fits and matches his/her application requirements. There are three ways to implement the organisation mechanism; either by integrating it within the agents themselves, designing the organisation externally to the agents, or a combination of the two [Dastani, 2008].

Property Name	Data Type	Description
Model Name	String	Unique indicative name of the institution.
Description	String	More details to describe this model.
List of Roles	Array	List of the member roles of this institution.
Role/Roles Relations	Array	List of role/role relations in this institution.
Roles Count	INT	The total number of institutional roles in this institution.
Verified	Boolean	Set to True once the model is verified otherwise it remains false.
Created on Date	Date	In the format ddmmyyyy.
Created on Time	Time	Time formatted as hhmmss.

Table 4.5: *MSMAS Institutional Roles Model Properties*

Designing the organisation mechanism external to the agents is normally motivated by the desire for an explicit organisational model, normative system or institution, where agents are designed to be regulated by a set of norms, that are used by the individual agents to decide how to behave or are enforced through monitoring and punishment mechanisms. It also permits independent verification and straightforward replacement when regulations (norms) need to be changed. In MSMAS we have chosen to support the concept of institutions explicitly, where the system designer can create a number of institutional roles and group them in one or more institutions. Each institution is presented by one institutional roles model. Table 4.5 shows the properties of each institutional roles model and Table 4.6 shows the properties of each institutional role.

In our approach we have chosen to implement the organisational mechanism explicitly to allow for separate management and transformation of our metamodel into any target programming language, whether that language provides programming constructs to implement the social concepts or not. Another advantage of our approach is that it allows any agent to play any role, as long as its individual goals are achieved through playing such role [van Riemsdijk et al., 2009].

An institutional role, in complex social settings, is distinguished from the system participant that plays the role. In the social organisation a person may cover multiple roles in his/her position. For example, an operations manager plans the workload of his unit and may or may not be the performance evaluator for his direct reports. This distinction between the person and the roles he may play is useful for separating intentional dependencies on a role from those on the system participant that plays the role. Under this separation of roles, we can easily control and identify conflicting roles, dependent roles, etc. Furthermore an organisation can

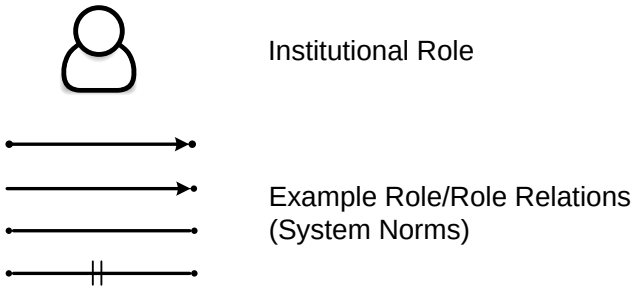


Figure 4-11: *MSMAS: Institutional Roles Model Notations*

be modelled as a set of roles related via dependencies, regardless of which individual system participants are playing those roles. Roles are also useful means for implementing security policies and monitoring and assessing social concepts such as trust and reputation [Giorgini et al., 2006, Liu et al., 2003].

Figure 4-11 shows the diagrammatic notations of the Institutional Roles Model while Figure 4-14 shows an abstract Institutional Roles Model involving three institutional roles (InstitutionalRole1, InstitutionalRole2, and InstitutionalRole3) and one institutional roles norm expressed as a relationship between InstitutionalRole1, and InstitutionalRole2. The full list of available norms to use within Invitational Roles Models are listed and discussed in Section 4.4.2. For a concrete example based on real-world scenario please refer to case study in Chapter 5 where Figure 5-11 and Figure 5-12 show the visual institutional roles models alongside the detailed properties of these models.

Property Name	**Data Type**	**Description**
Role ID	INT	Auto incremented system generated ID.
Role Name	String	Unique Name.
Description	String	More details to describe this institutional role.
Member of Institution	Array	List of institutions that this role is member of.
Role/Role Relations	Array	List of system norms of the type role/role relations that are set on this role.
Played By	Array	List of system participants that are assigned this role.
Responsible for Activity	Array	List all business activities that this role takes part in.

Table 4.6: *Institutional Role Properties in MSMAS*

4.4.2 Norms of MSMAS Institutional Roles

Specifying an organisation can be done through specifying the inter-agent relationships that exist within this organisation [Zambonelli et al., 2001]. MSMAS adopts an explicit statement of the society's organisational structure, where the system is organised in a number of institutions each of which has an associated finite set of roles, the system participants can play one or more of these roles in one or many system institutions. In MSMAS, these inter-relationship specifications are centered around the abstract roles the system participants can play and how these roles relate to each other. Role specification allows for defining behaviour patterns in an abstract way, independent from each individual system participant. In this sense, roles are considered as system participant types, so when a system participant takes part in an institution and plays one of the institution's roles, it should conform to that pattern of behaviour. All system agents that adopt the same role are normally granted the same rights and duties, and are expected to obey the same restrictions applied to that role.

An important aspect of specifying the organisation structure is to specify the organisational rules. These are simply the relationships and constraints between roles, between interaction protocols and between roles and interaction protocols [Zambonelli et al., 2001]. Although some modelling efforts have been directed at specifying the MAS organisation, such as the work of Ferber and Gutknecht [1997] and Wooldridge et al. [2000b], their model of organisation was as a collection of roles that lacked the specification of the organisational rules. Zambonelli et al. [2003b] and Zambonelli et al. [2001] state that organisational rules can be expressed as global constraints prescribing the behaviour of the members of an organisation and can be formalised by one of two approaches (i) a subset of a first-order temporal logic [Manna and Pnueli, 1992], or (ii) regular expressions [McNaughton and Yamada, 1960]. One of the most common relations in real world organisations is the Hierarchical Role (CPR) which specifies that one of a pair of roles has authority over the other role. This type of role/role relation is required to address the problem of how to express authority/control in an organisation. The semantics of authority in real life can vary widely based on the application domain (e.g. authority in military settings is quite different from authority in the context of academic article review process). In the military, this is a clear implementation of the notion of "command and control" [Zambonelli et al., 2001], where the lower rank person must obey and execute the commands of a higher-rank person, while in article review scenarios the article author might change his article based on the recommendations received from the article reviewer or just ignore all/part of them. Hierarchical role modelling in MAS has been studied by many researchers such as Zambonelli et al. [2001], Esteva et al. [2001] and Grossi et al. [2005]. We take a simplified view by providing the notion of child/parent role/role relation and we leave expressing and handling of the semantics of this relation to the implementation at the application level. Properties such as Transitive [2] and Anti-symmetric [3] should be defined and

[2] **Transitive**: if $(r_a, r_b) \in C$ and $(r_b, r_c) \in C$ then $(r_a, r_c) \in C$.
[3] **Anti-symmetric**: if $(r_a, r_b) \in C$ then $(r_b, r_a) \notin C$.

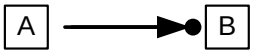
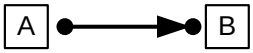
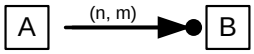
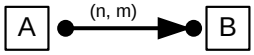
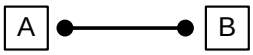
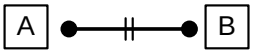
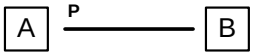
ConDec++ Notation	ConDec++ Visual notation	MSMAS Role/Role Relation
Precedence Relationship: If ***B*** Is performed ***A*** should have been performed before it		Sequential Roles: The system participant has to play ***Role B*** after playing ***Role A***
Succession Relationship: every execution of ***A*** should be followed by the execution of ***B*** and each ***B*** should be preceded by ***A***		Joint Roles: ***Role B*** must be played after playing ***Role A,*** and ***Role A*** must be played before playing ***Role B***
Precedence Relationship with Time Constraint		Sequential Roles: The system participant has to play ***Role B*** after minimum delay of n and maximum delay of m after playing ***Role A***
Succession Relationship with Time Constraint		Joint Roles: The system participant must play ***Role B*** after minimum delay of *n* and maximum delay of *m* of playing ***Role A,*** and it must have played ***Role A*** in order to play ***Role B***
Coexistence Relationship: If either ***A*** or ***B*** is performed, the other one has to be executed as well.		Coupled Roles: The system participant has to play both ***Role A*** and ***Role B***
Not Coexistence Relationship: If one of ***A*** or ***B*** is performed, the other one can not be executed.		Disjoint Roles: If the system participant plays ***Role A*** then ***Role B*** can not be played, and vice versa.
No constraint		Amicable Roles: The system participant can play any or both of ***Role A*** and ***Role B*** without any restriction
No constraint		Child/Parent Roles: The system participant that plays ***Role B*** *may* report to any system participant that plays ***Role A,*** and the system participant that plays ***Role A*** may delegate to other system participants that play ***Role B***

Figure 4-12: *ConDec^{++} Notation and its Mapping to MSMAS Role/Role relation concepts*

interpreted according to the application domain.

Figure 4-12 visual notation and and semantic of our defined role/role relations.

- **Role/Role Relations Set Through Institutional Roles Models**
  - **Sequential Roles (**SR**)**: these are the pairs of roles where the system participant is required to play the first role before it can play the second role. Sequential Role/Role Relations can also be defined with a time constraint as shown in Figure 4-13, which expresses that role b can be played only inside the time window ranging from n to m time units after the system participant plays role a. This means playing role b has to happen between a minimum time delay of n and maximum time delay of m time units.
  - **Joint Roles (**JR**)**: these are pairs of roles where the system participant is required to play both, but one after another in a specified order, or neither. An example is the

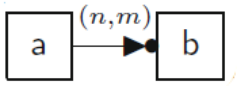


Figure 4-13: *ConDec^{++} Notation for Sequential Relation with Time Constraint*

requirement in a marketplace that a type of seller should fulfill their customers' orders themselves. The agent has to it has to play the role of Shipper after playing the role of Seller. Joint Role/Role Relation can also be defined with a time constraint in the same manner described in Sequential Role/Role Relation.

- **Coupled Roles** (CR): these are pairs of roles that are coupled together where the system participant is required to play both or none, but in either order: once one of them is played the other one has to be played. Notice that there is no need to specify time constraint on Coupled Role/Role relation as it is an undirected relation with no sequence defined in any specific direction.
- **Disjoint Roles** (DR): these are mutually exclusive roles, where only one of them can be played by a system participant, and once the system participant plays any of them it cannot play the other role. An example of this is when you have a Coder and Code reviewer, with a requirement that states that an individual shall not review their own code (*four eyes principle*). There is no need to specify time constraint on this relation.
- **Amicable Roles** (AR): these are the pairs of roles where the system participant can play one or many of them at the same time without raising any conflict.
- **Child/Parent Roles** (CPR): this is a relationship between two roles, where one of them has authority over the other. Authority, in social science, is a legitimate or socially approved use of power which one person or a group holds over another. The element of legitimacy is the main property that distinguishes authority from the more general concept of power. While power can be enforced, Authority depends on the acceptance by subordinates of the right of those above them to give them orders. In MSMAS Authority grants the parent role the right to delegate tasks to its children roles. MSMAS does not specify how to implement this relation, we just offer the system norm to express it and it is up to the system developer to decide on best implementation according to this/her application domain.

The set of all Institutional Roles Relations IR is then presented as: $IR_{msmas} = \langle SR \cup JR \cup CR \cup DR \cup AR \cup CPR \rangle$ and each role set in turn is defined as: $R_{inst} = \langle inst, R, R_{rel} \rangle$ where R is the set of roles that belong to institution $inst$ and R_{rel} is the set of relations between these roles in R.

The set of all Institutional Roles IR is then presented as: $IR_{msmas} = \langle SR \cup JR \cup CR \cup DR \cup AR \cup CPR \rangle$ Each role set in turn is defined as: $R_{inst} = \langle inst, R, R_{rel} \rangle$ where R is the set of roles that belong to institution $inst$ and R_{rel} is the set of relations between these roles in R.

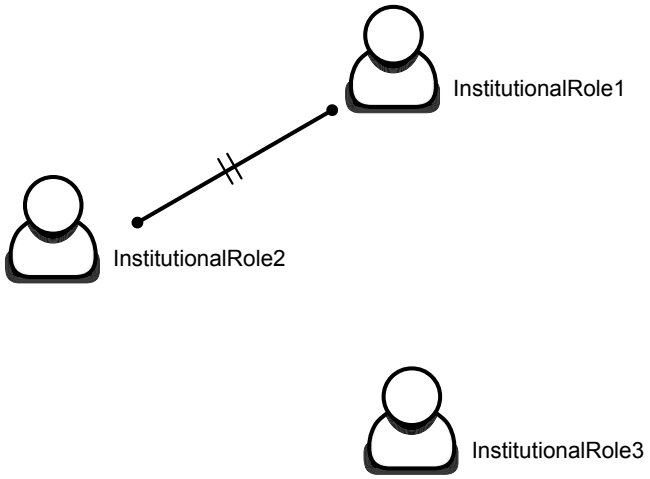


Figure 4-14: *MSMAS: An Abstract Institutional Roles Model*

Once the system designer has created all institutional roles models that satisfy the organisational structure, he/she might choose to create system participants and assign them the appropriate institutional roles. Or he/she can move on to the next step, which is refining the system goals through setting goal/goal system norms as discussed in the following section.

4.4.3 Composite Business Processes Models

In the first phase, the user creates a system goals model, but that model is just a tree of all system goals and it does not explain if all sub-goals of a goal are needed to achieve their super goal or some subset of them is sufficient. The sub goals of a goal should be seen as a disjunction of conjunctions. For example, if we have a system goal g which has a set of subgoals G and g can be achieved by either the set of subgoals $G_1 = (g1, g2, g5)$ or another set of subgoals $G_2 = (g1, g3, g4)$ then $g \models (g1 \wedge g2 \wedge g5) \vee (g1 \wedge g3 \wedge g4)$.

To specify such relations between the system goals, we can use the composite business process models, which are representations of the composite system goals to set one or more goal/goal relations between the subgoals.

Table 4.7 shows the set of properties each Composite Business Process Model has, and Table 4.8 show the properties of each Business Process.

Property Name	**Data Type**	**Description**
Model Name	String	Unique indicative name of this Composite Business Process and normally indicative of the Composite System Goal it presents.
Description	String	More details to describe this model.
Sub Business Processes	List	A list of all sub business processes in this model.
		Continued on next page

Table 4.7 – continued from previous page

Property Name	Data Type	Description
Goal/Goal Relations	List	A list of all goal/goal relations used in this model.
Sub Business Processes Count	INT	The total number of business processes within this model.
Verified	Boolean	Set to True once the model is verified otherwise it remains false.
Created on Date	Date	In the format ddmmyyyy.
Created on Time	Time	Time formatted as hhmmss.

Table 4.7: *Composite Business Process Model Properties*

Property Name	Data Type	Description
Business Process Name	String	Unique indicative name of this Business Process and normally indicative of the System Goal it presents.
BP ID	INT	Unique ID.
Type	String	The Type of the Business Process (Composite - Basic).
Fulfills Goal	String	The name of the system goal associated with this BP.
Description	String	More details to describe this model.
Super Business Processes	List	A list of all super business processes.
Goal/Goal Relations	List	A list of all goal/goal relations set between this BP and other BPs.

Table 4.8: *Business Process Properties*

4.4.4 Norms for MSMAS System Goals

System Goals in MSMAS are represented and specified in various forms, in this section we briefly list these forms and then explain how the user can set the system norms within each presentation form.

The first representation is a system goals tree (System Goals Model) . In this form the user breaks down each system goal into its sub-goals and and specifies the parent-child relation between the Super Goal and the set of its Sub-Goals. In the System Goals Model, each Goal must have at least one parent goal, the only exception is the General System Goal, which is the root class of system goals. In MSMAS there are two other system goal types: Composite System Goal and Basic System Goal, the first must have at least one Sub-Goal, Composite System Goals cannot be the goal tree leaves, while Basic System Goals cannot have any Sub-Goals. In the System Goals Model there is no limit on how many Parents or how many Children any goal can have. Any Composite Goal might have any number of Parent Composite Goals,

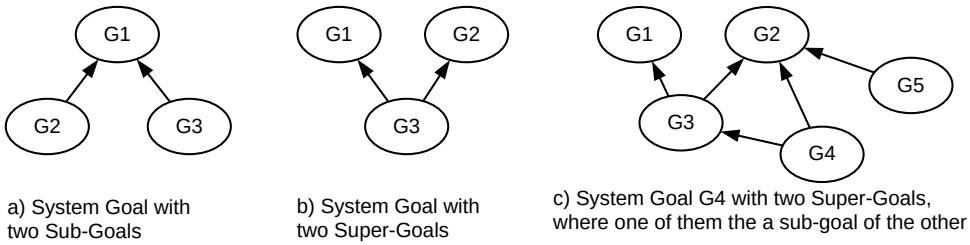


Figure 4-15: *Possible System Goal Structures in MSMAS System Goals Model*

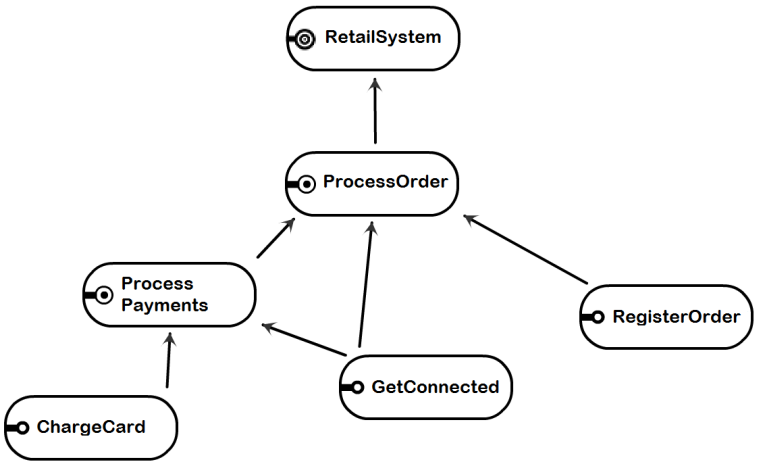


Figure 4-16: *Example System Goals Model with multi-level structure of System Goals*

and can have any number of Composite and/or Basic system Goals. There is no concept of level within the goals tree in the System Goals Model.

Figure 4-15 shows a number of possible system goal structures, where in sub-figure *a* a goal (G1) has two subgoals (G2, G3), in sub-figure (*b*) goal (G3) has two super-goals/parentgoals (G1, G2), and in sub-figure (*c*) goal (G4) has two super-goals/parentgoals (G2, G3) where G2 is a super-goal of (G3, G4). MSMAS supports these different structures to help system designers to model all different use cases. An example that shows how case (C) is a valid use case is shown in Figure 4-16, where in this system fragment there is an order processing process, where the goal is to process the customer's order (PrcosessOrder). This goal requires to register the customer order (RegisterOrder) before or while processing the payments. To Register an order the system is required to obtain a connection to the data store (GetConnected). The system designer chooses to make GetConnected an independent function to allow for reuse. In this particular example the designer knows that current payments integration would require obtaining a connection as well, but future payments integration might be self-contained. So the system designer sets GetConnected as a subgoal of (ProcessPayments) as well as its super goal (ProcessOrder).

Building the system goals tree is an important design decision that will drive other MSMAS models, so it is important for the system designer to understand how to achieve a big goal

through the achievement of its smaller sub-goals, and the process of goals breakdown. Goal-analysis and formation is well studied and commonly-used technique to support requirements elicitation, requirements negotiation, requirements specifications and requirements validation. In this context we are interested in requirements specifications, which is about relating business goals to functional and non-functional system components. This relationship is addressed in terms of three broad categories: *goal elaboration*, *scenario definition* and *non-functional requirements definition*. The work of Anton [1996] and Antón et al. [2000] offers accessible guidelines on how to extract goals into one ordered goal set, furthermore, how to create a goal schema for the operationalised goals, the responsible agents, scenarios and obstacles. Cockburn [2000] suggests the use of goals to structure scenarios by connecting every action in a scenario to a goal assigned to an actor. Similarly, Rolland et al. [1998] proposes the organisation of scenarios using goal hierarchies. MSMAS recognises and defines these goal/goal relations and offers formal representation of the goals tree.

- **Defining Goal/Goal relations within the System Goals Model**
  - **Child/Parent Goal/Goal Relation (CPG):** Is a relation between two system goals which states that one goal is a subgoal of the other goal (parentgoal/super goal). This means achieving the subgoal contributes to the achievement of the parentgoal/super goal. From the parentgoal/super goal perspective this means to achieve the parentgoal/super goal the system participant needs to achieve a set of subgoals, that might include this subgoal.

In MAS where the system might be an open environment, the system goals could complement each other at a point of time, then a change in the environment could make these same goals irrelevant or in some cases conflicting goals. This particular situation becomes obvious in the case of self-interested agents where their behaviour and attempts to achieve some goals that suit their personal beliefs might be contradictory or undesired from a system-holistic point of view.

To address this issue MSMAS allows the system designer to represent this requirement by setting constraints at the system goals level that could enable the discovery of any potential or arising conflicting situation. Once a conflict of goals, or undesired course of actions, is detected a corrective action can be applied to resolve the current conflict. Although ConDec^{++} defines a wide range of relations, we limit goal/goal relations to only small set which in our view covers most uses cases. This set includes the five types of Goal/Goal relations as shown in Figure 4-17 with ConDec^{++} visual notation. The details of this goal/goal types as follows:

- **Defining Goal/Goal Relations within Composite Business Process Models**
  - **Sequential Goals (SG)**: these are the pairs of goals where one goal cannot be achieved before another goal has already been achieved. Sequential Goal/Goal Relation can also be defined with time constraints such as the example shown in Figure 4-13 which expresses that goal b is achievable only inside the time window ranging from n to m time units after each achievement of goal a. This means

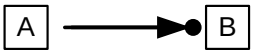
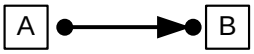
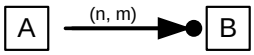
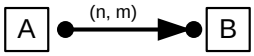
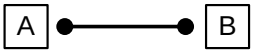
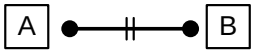
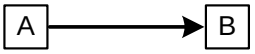
ConDec++ Notation	ConDec++ Visual notation	MSMAS Goal/Goal Relation
Precedence Relationship: If ***B*** Is performed ***A*** should have been performed before it		Sequential Goals: ***Goal B*** has to be achieved only after achieving ***Goal A***
Succession Relationship: every execution of ***A*** should be followed by the execution of ***B*** and each ***B*** should be preceded by ***A***		Joint Goals: ***Goal B*** must be achieved after achieving ***Goal A,*** and ***Goal A*** must be achieved before achieving ***Goal B***
Precedence Relationship with Time Constraint		Sequential Goals: The system participant has to achieve ***Goal B*** after minimum delay of n and maximum delay of m after achieving ***goal A***
Succession Relationship with Time Constraint		Joint Goals: The system participant must play ***Role B*** after minimum delay of *n* and maximum delay of *m* of playing ***Role A,*** and must have played ***Role A*** in order to play ***Role B***
Coexistence Relationship: If either ***A*** or ***B*** is performed, the other one has to be executed as well.		Coupled Goals: Both ***Goal A*** and ***Goal B*** must be achieved
Not Coexistence Relationship: If one of ***A*** or ***B*** is performed, the other one can not be executed.		Disjoint Goals: If ***Goal A*** has been achieved then ***Goal B*** can not be achieved***,*** and vice versa.
No constraint		Amicable Goals: Any or both of ***Goal A*** and ***Goal B*** can be achieved without any restriction
No constraint		Child/Parent Goals: ***Goal A*** is a Parent-goal of ***Goal B,*** and ***Goal B*** is a child-goal of ***Goal A***

Figure 4-17: *ConDec^{++} Notation and its Mapping to MSMAS Goal/Goal relation concepts*

the achievement of goal b has to happen between a minimum time delay of n and maximum time delay of m time units.

- **Joint Goals** (JG): these are pairs of goals where both are required to be achieved but one after another in a specified order, or neither. Joint Goal/Goal Relation also be defined with time constraint in the same manner described of previous Sequential Goal/Goal Relation.
- **Coupled Goals** (CF): these are pairs of goals that are coupled together, where they both have to be achieved or none, but in either order: once one of them is achieved the other one has to be achieved. Notice that there is no need to specify time constraint on Coupled Goal/Goal relation as it is an undirected relation with no sequence defined in any specific direction.
- **Disjoint Goals** (DF): these are mutually exclusive goals, where if one of them has been achieved then the other goal should not be achieved. Notice that there

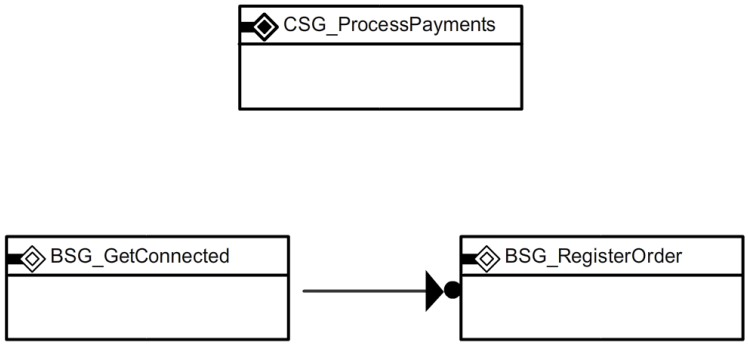


Figure 4-18: *Example Composite BP Model "Process_Order_CBP" with a Goal/Goal relation*

is no need to specify a time constraint on a Disjoint Goal/Goal relation as it is an undirected relation with no sequence defined in any direction.

- **Amicable Goals (**AG**)**: these are the pairs of goals where any one or many of them can be achieved at the same time without raising any conflict.

Goal/Goal relations can be defined at ***the second presentation of system goals (Composite Business Process Models)***: each Composite Business Process (CBP) model represents one of the System Composite Goals and each sub-process within this model corresponds to one subgoal. The system designer can define any number of non-conflicting Goal/Goal relations between the Business Processes corresponding to the goal pair.

Figure 4-18 shows the Composite Business Process Model of "Process_Order_CBP" that represent the Composite System Goal "Process Order CSG". As shown in Figure 4-16 the CBG "Process Order" has three sub-goals (ProcessPayments), (GetConnected) and (RegisterOrder). The system designer sets a sequential goal relation that represents a goal norm between (GetConnected) and (RegisterOrder), which defines the sequence for achieving these two goals to avoid a failed attempt to register an order before obtaining a connection.

MSMAS Goal Norms (goal/goal relations) are formalised by means of the CLIMB language as detailed in Chapter 6, Section 6.3, where we show the mapping of each relation and how to reason about them.
The set of all system goal relations G is then presented as:

$$G_{msmas} = \langle SG \cup JG \cup CG \cup DG \cup AG \cup CPG \rangle$$

Each goal set of any system participant in turn is defined as: $G_{sp} = \langle SP, G, G_{rel} \rangle$ where G_{sp} is the set of goals of a system participant *(sp)* and G is pairs of goals and G_{rel} is the set of relations between these goals in G. This set dynamically changes according to which roles the SP plays or drops.

Completing the institutional roles models and the refinement of the system goals marks the end of the initial design stage and we can move on to the second stage of the second phase (Detailed System Design Phase) which is covered in the following sections.

4.5 System Design Phase: Detailed Design Stage

During initial design stage we created the system skeleton, where we identified the system goal relations and specified the main organisational structure in the form of institutions and institutional roles that agents can play to achieve the system goals. Each composite system goal has a composite business process model to allow us to specify any system norms at the goal level. Meanwhile each basic system goal has a basic business process model. The detailed system design stage is about fleshing out the system details, focusing on creating the actual business activities that form each basic business process. Each activity in the basic business model is an atom, and it is associated with one or more institutional roles. The execution of a set activities would normally lead to the achievement of the basic goal presented by that basic business process. If two or more agents are responsible for the execution of an activity, they need to interact. The specification of these agents' interactions as expressed in the form of communication protocols. By the end of this stage the system modeller achieves:

- Creation of the full set of *Basic Business Process Models* through specifying the business activities, and definition of the *entry* and *exit* activities of each model.
- Specification of further activities for an alternative execution routes if required.
- Specification of system norms at the business activity level.
- A set of custom communication protocols, if required, for the application.
- Finalisation of each Basic Role/Activity View Model by setting a communication protocol between each business activities pair if required.

The following sections go through these tasks in detail, and use example model descriptions, and examples based on the our case study.

4.5.1 Basic Business Process Models

Business process modelling and management is discussed in some detail in Section 2.5, but we recall the definition of a business process by Hammer and Champy [1993] which has influenced the development that follows. They defined the business process as "a collection of activities that take one or more kinds of input and create an output that is of value to the customer".

This definition matches our view in part. It is good that it highlights the importance of the inputs (preconditions or triggers) and the outputs (postconditions or new system state) and the use of a collection does not imply that the set of activity is an ordered list. This abstract definition of activities suits the nature of MAS, however the definition is incomplete as the execution route of the business process activities in some cases has to be constrained. To address this issue Weske [2012] defines the business process as "A business process consists of a set of activities that are performed in coordination in an organisational and technical environment. These activities jointly realise a business goal...". The use of coordination implies that there is a degree of interaction between the activities and a logical line that defines the execution path. Also this definition defines the purpose of the business activity through grouping the activities

under the achievement of a business goal.

In MSMAS we define a Business Process as a set of activities associated with a set of organisational roles that are responsible for their execution, each activity contributes to the achievement of a defined system goal by changing the system state in the form of a post condition (belief). Each activity may or may not have a set of preconditions that trigger the execution of that activity once satisfied.

Our definition emphasises the fact that business activities are linked to institutional roles and they are triggered when their pre-conditions have become true. The scope of the pre-conditions here is wide open; they can be internal conditions such as a private belief of the system participant that plays the role responsible for this activity, or global pre-conditions that are global states of the system or an activity/activity relation that defines order of execution or existence relationship between these activities.

In MSMAS, each activity is considered as a primitive business plan, that forms one step in one or many possible system plans presenting all possible execution routes, and each leads to the achievement of the system goal associated with that business process. We adopt the declarative modelling approach when modelling the relations between business activities.

The design of the basic business process model is about defining each business activity and linking it with the appropriate roles. The model consists of the details of one or more business activities and one or more activity/activity relations, we use the ConDec^{++} language and graphical notation[4] for these relations. Each Basic Business Process Model has the properties shown in Table 4.9. When designing a basic business process in MSMAS, we require the definition of an *entry* activity and an *exit* activity. The entry activity is the first activity of that business process and normally has the preconditions of that business process, while the exit activity is the one that marks the successful execution of the business process, and its post conditions are normally the achievement of the system goal associated with this business process. It is allowed for an activity to be the entry as well as the exit activity, but this would indicate that this business process has only one business activity. It is also possible to have multiple exit activities in which case each would have different pre-conditions.

In the basic business process example, we illustrate the use of *overloading* concept where different activities with the same name are invoked based on the data types of the parameters passed. They do not necessarily have the same post conditions.

Property Name	**Data Type**	**Description**
Model Name	String	Unique indicative name of this Basic Business Process and normally indicative of the Basic System Goal it presents.
Description	String	More details to describe this model.
Business Activities	List	A list of all business activities in this basic business process model.

Continued on next page

[4]MSMAS designer tool does not yet support the full set, however this is part of the future improvement plans

Table 4.9 – continued from previous page

Property Name	Data Type	Description
Activity/Activity Relations	List	A list of all activity/activity relations used in this model.
Business Activity Count	INT	The total number of business activities within this model.
Verified	Boolean	Set to True once the model is verified otherwise it remains false.
Created on Date	Date	In the format ddmmyyyy.
Created on Time	Time	Time formatted as hhmmss.

Table 4.9: *Basic Business Process Model Properties*

And each business activity has the following properties as shown in Table 4.10.

Property Name	Data Type	Description
Business Activity Name	String	Unique indicative name of this Business Activity and normally indicative of the Basic System Goal it presents.
Business Activity ID	INT	Unique ID
Description	String	More details to describe this model.
Responsibility of	List	A list of all institutional roles that are responsible for the execution of this business activity.
Activity/Activity Relations	List	A list of all activity/activity relations set on this activity.
Required Communication protocols	List	A list of all communication protocols used between the system participants to execute this activity.
Partially Fulfills Goal	String	The name of the basic system goal associated with the basic business process where this business activity is part of.
Pre-Conditions	List	A list of beliefs required to be true before this business activity can be executed.
Post-Conditions	List	A list of beliefs that become true once this business activity is successfully executed.
Parent Business Process	String	The name of the basic business process where this business activity is part of.

Table 4.10: *Basic Activity Properties*

Figure 4-19 shows the diagrammatic notations of Basic Business Process Model while Figure 4-20 shows an abstract basic business process visual model that contains three business activities (BA_BusinessActivity1, BA_BusinessActivity2, and BA_BusinessActivity3) and one activity/activty norm between BA_BusinessActivity1 and BA_BusinessActivity3 where BA_BusinessActivity1 is an entry activity with no precondistions and BA_BusinessActivity3

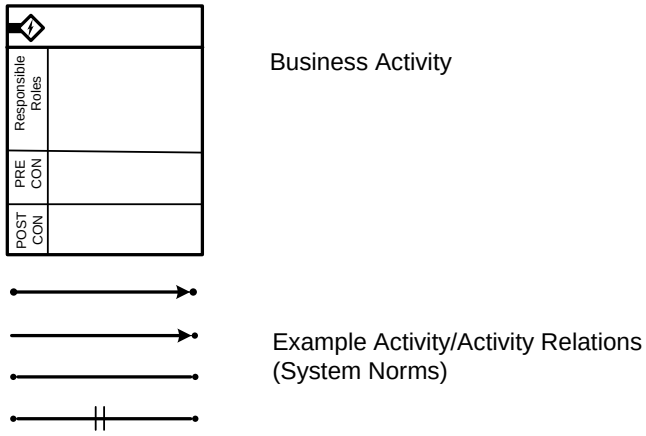


Figure 4-19: *MSMAS: Basic Business Process Model Notations*

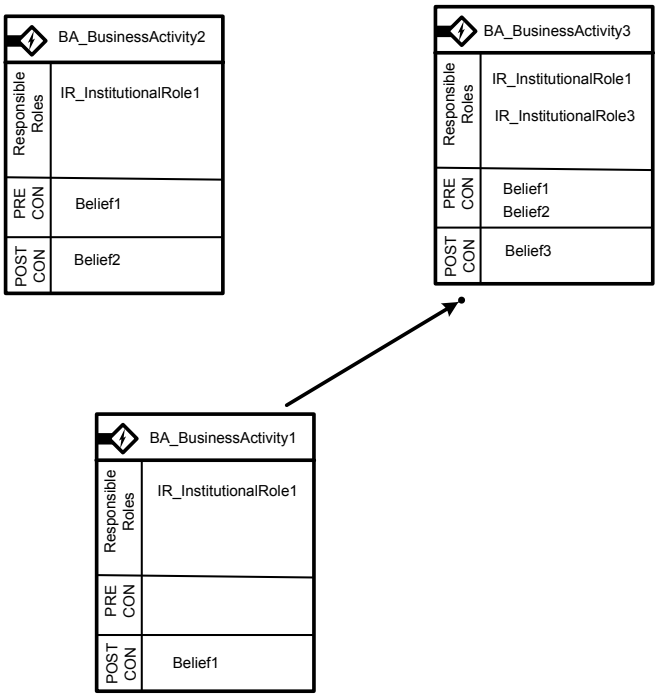


Figure 4-20: *An Abstract Basic Business Process Visual Model*

is the exit activity. For a real world example refer to Figure 5-14 Page 179 that shows the basic business process model of the basic business process (BBP_PublishSupplierStock).

4.5.2 Norms of MSMAS Business Activities

In MSMAS only the Basic Business Process Model (BBPM) contains the detailed specifications of the actual business activities that lead to the achievement of one Basic System Goal (BSG). Each BSG can be achieved through the execution of one or more activities. MSMAS allows designers to assign any ConDec^{++} constraint to any business activity and any ConDec^{++}

relation to any business activity pair within the BBPM. The full set of relation formulae and their notation as well as their CLIMB mapping can be found in [Montali, 2010], but a few examples of these possible relations are shown in Figure 4-3, while the full list can be found in Appendix F.

In MSMAS BBPM, each Business Activity has a number of post conditions that can be considered as a subgoal of the basic system goal associated with this BBPM. MSMAS requires the system designer to specify one ***Entry*** activity of this business process, and one ***Exit*** activity. These are special kinds of activities that mark the start of and the completion of the basic system goal associated with this basic business process. The event associated with the start of an entry activity triggers the event of achieve_goal of the basic system goal, and the event of completion the exit activity triggers the event satisfy_goal. This is explored in more detail in Chapter 6 Section 6.3.

4.5.3 Communication Protocols

Communication protocols are standard patterns or uniform methods of information exchange between two or more system participants. They specify the behaviour of a system at the communication level. Communication protocols are useful for identifying, in the event of failure, which system participant has failed, or violated the expected behaviour pattern. The system in question can utilise this through a mechanism such as static analysis of the execution traces or run-time monitoring. Both are dependent on checking happened events against a set of expected actions.

Exchanging information and knowledge between system participants requires a shared, reusable communication language that allows a system participant to share a common syntax, semantics, and pragmatics [Finin et al., 1994]. Thus, the communication problem as a major enabler for autonomy has three aspects:

- **Syntax:** is about the structure of the message or collection of messages;
- **Semantics:** what is the meaning of communication symbols or specific structure in terms of logics or meanings;
- **Pragmatics:** is about how the symbols are interpreted and how they are used to initiate and progress in the communication act.

The Speech Act Theory (SAT) introduced by Austen [1962] and developed further by Searle [1969], then Colombetti and Verdicchio [2002], is the base block that all agent communication languages depend on. As stated by Fasli [2007]; “Perhaps the most influential theory in agent communication is that of speech act”. There are few standard agent communication languages such as Knowledge Query Manipulation Language (KQML) [Finin et al., 1994, Labrou and Finin, 1998], Agent communication Language (ACL) [FIPA, 2008], and Knowledge Interchange Format (KIF) [Genesereth et al., 1992], however the most established standard is the FIPA Agent communication language (FIPA-ACL).

MSMAS allows for the use of either user-defined Custom Messages, user-defined Custom

Communication Protocol, FIPA Message, or FIPA Interaction Protocol. Both FIPA messages and protocols are part of FIPA Agent Communication Language (FIPA-ACL)[5]. The main reason that motivated us to allow the user to design custom messages and protocols is that MSMAS has four types of system participants Agent, Human Actor, Service and Environment and while FIPA ACL was built specifically to support agents communication, it does not scale well enough to be able to support different classes of system participants. For example with regard to cardinality FIPA assumes that typical interaction happens between pairs of individual agents, although some FIPA protocols involve mediators and multiple parties[6] and FIPA lacks communicative acts that support multiple heterogeneous types of agent communication to enable the exchange of knowledge, mental attitudes, human interaction, non-agent computation and network interaction [7]. Another example is when the service has an API: then the message syntax and how they are exchanged is likely to differ from the typical FIPA ACL. MSMAS aims to be expandable to support different scenarios, hence the motivation for us to support custom formats as well as FIPA ACL.

In MSMAS, a Communication Protocol (CP) is defined as a set of Communication Messages (CM) that facilitate the interaction between one or more system participant. CPs are required for effective communications and will vary according to the situation and the type of activity they are used to facilitate. CMs are of many types and each CM has multiple parameters. FIPA ACL messages have only one mandatory parameter, namely *performative*, but usually ACL messages will also contain sender, receiver and content parameter as well. More details on FIPA ACL is given in Appendix B.

- **Custom Communication Message:** In MSMAS, we allow the system designer to specify custom messages with any different syntax given that the message have the set of properties shown in Table 4.12.
- **Custom Communication Protocol:** In MSMAS, we allow the system designer to specify custom communication protocols through the creation of custom protocol models. Each communication protocol has a set of custom communication messages, and a set of message/message relations that specify the system norms that have to be observed while communicating using this particular protocol. Communication protocol models have the set of properties as shown in Table 4.11 and each communication message has the list of properties as shown in Table 4.12.
- **FIPA Interaction Messages:** FIPA defines a number of standard communicative acts where the agent sends a message of specific type and format. A summary of all identified messages is listed in Table B.1 in Appendix B.
- **FIPA Interaction Protocols:** FIPA-ACL is a language for agent to agent communications by means of communicative acts. Within a communicative act an agent is exchang-

[5] FIPA: `www.fipa.org`

[6] E.g. Contract Net Task-sharing Protocol

[7] Highlighted by Stefan Poslad in Review of FIPA Specifications: `http://www.fipa.org/subgroups/ROFS-SG-docs/ROFS-Doc.pdf`, retrieved 20 January 2014

ing message with another in a particular manner. FIPA has a number of protocols and has defined the properties of its messages. For more details please refer to Appendix B. In MSMAS, we allow the system designer to specify that an activity requires the use of a FIPA protocol and to specify which institutional role is the initiator of this protocol and which institutional role is the receiver. The system designer can also just specify that there is only one message used in a particular communication interaction. All FIPA message types are available for use in specifications.

Property Name	**Data Type**	**Description**
Model Name / CP Name	String	Unique indicative name of the communication protocol.
Communication Protocol ID	INT	Unique ID
Description	String	More details to describe this protocol.
Used in	List	A list of business activities that require the use of this communication protocol.
Used by	List	A list of all institutional roles that use this protocol.
Message IDs	List	A list of all Communication messages that are part of this model. Formally this list is an ordered list of disjunction sets of message conjuncts .
Communication Protocol Type	String	Either FIPA or Custom.
Message/Message Relations	Array	List of message/message relations in this communication protocol.
Messages Count	INT	The total number of messages in this communication protocol.
Created on Date	Date	In the format ddmmyyyy.
Created on Time	Time	Time formatted as hhmmss.

Table 4.11: *Communication Protocol Properties in MSMAS*

Figure 4-21 shows the diagrammatic notations of Communication Protocol Model while, Figure H-1 shows the custom communication protocol CCP_TranslateFile from our case study, which facilitates the interaction between two system participants the first one plays the IR_SupplierAgent institutional role and the other is playing IR_TranslationService institutional role. The protocol has six messages and six message/message relations. All possible execution routes of this protocol are shown in the AMUL diagram shown in Figure 5-7, while Table 5.11 shows its descriptor. This protocol is to be used by the supplier agent that needs to translate its custom file into the SSLF standard format. SSLF is the only format recognised by the central stock manager and Supplier agents have to use it to be able to submit their updates. In this protocol, *CP_TranslateFile*, there are a maximum of four steps and a minimum of two. As shown in Figure 5-7, the communicative act starts by the supplier agent sending a *Request* message. This

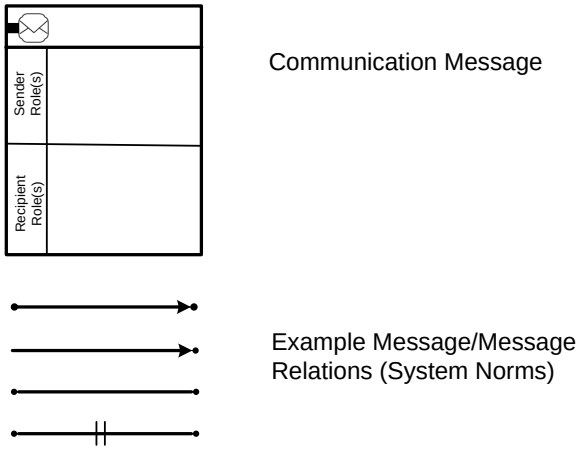


Figure 4-21: *MSMAS: Communication Protocol Model Notations*

message could be designed to allow the supplier agent to specify the file format, the size, and number of records. The use of a *Request* message requires the translation service to respond, it has one of three options. The first is to *Agree* to the translation request, in this case the supplier agent after receiving the *Agree* message reply back with an *Inform* message that contain its file, then the translation service replies either with the file translated attached to an *Inform* message or with an *Error* message that contains a failure code indicative of the type of error it has encountered. The second option in step two can be to *Refuse* the translation request. This could be due to any reason such as if the service is too busy or it is not qualified to translate this particular custom file format. The third option in step two is to send an Error message, that normally indicates a failure and it contains a failure code.

MSMAS allows the use of message/message relations, which are a type of system norm at the communication level. Message/message relations are listed in Section 4.5.4

Please note that creating a communication protocol does not require the specification of the associated institutional roles: these can be assigned at later stage during the creation of Basic Roles/Activity Models. Creating custom individual messages is also possible, where multiple institutional role can use them.

Figure 4-22 shows MSMAS visual communication protocol model of FIPA Subscribe Interaction Protocol, for the explanation of the protocol please refer to Appendix B.

Property Name	**Data Type**	**Description**
Message Name	String	Optional name for the message.
Message ID	INT	Unique ID
Description	String	More details to describe this message.
Used in activity	List	A list of all business activities that require the use of this message.
Used by	List	A list of all institutional roles that use this message.

Continued on next page

Table 4.12 – continued from previous page

Property Name	Data Type	Description
Message/Message Relations	Array	List of system norms of the type message/message relations that are associated with this message.
Sender	List	A list of all names of potential institutional roles that can be a sender of this message.
Receiver	List	A list of all names of potential institutional roles that can be a receiver of this message.
Reply to	String	This parameter indicates that subsequent messages in this conversation thread are to be directed to the agent named in the reply-to parameter, instead of to the agent named in the sender parameter.
Content	ANY	The content of the message as an object of predefined types such as FILE, String, INT ...etc.
Language	String	The the language in which the content parameter is expressed such as (FIPA or CUSTOM).
Member of Protocol	List	List of all IDs/Names of the the interaction protocol that the sending agent is employing with this message.
Conversation-id	INT	A conversation identifier which is used to identify the ongoing sequence of communicative acts that together form a conversation. This is useful to differentiate between the concurrent interactions using the same protocols
Reply-with	String	Introduces an expression that will be used by the responding agent to identify this message.
In-reply-to	String	An expression that references an earlier action to which this message is a reply.
Reply-by	String	A time and/or date expression which indicates the latest time by which the sending agent would like to receive a reply

Table 4.12: *Communication Message Properties in MSMAS*

4.5.4 Norms of MSMAS Communication Protocols

In open systems where a set of autonomous components interact it is not possible to make assumptions neither about the internal structure nor the real intentions of the interacting parties. Interacting agents need to plan their future communicative actions relying on a common semantics of the norms that govern the exchanged messages. The design and most implementations of FIPA do not take into account the normative consequences of message exchanges, which makes it hard to verify if the communicative acts under specified conditions are compliant with the expected actions as specified. Many researchers have proposed to treat communicative acts in terms of commitments and to monitor their state on the basis of the agents' actions such as

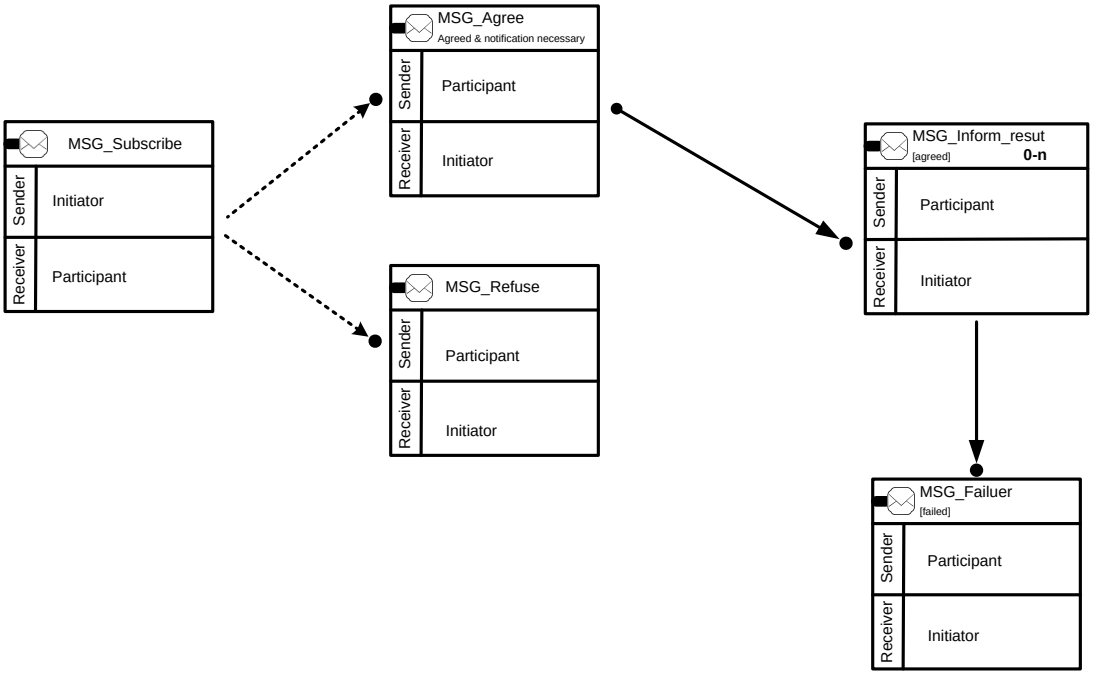


Figure 4-22: *Visual Representation of FIPA ACL "Subscribe Interaction Protocol"*

the work of Colombetti [2000], Singh [2000] and Baldoni et al. [2013].

A communication protocol in MSMAS is a set of communication messages interchanged between a number of system participants playing particular institutional roles. MSMAS adopts an explicit approach of defining the possible sequence of a message exchange by expressing the communication protocol as number of communication messages, where each message is considered an activity in a business process. We express the specification of execution paths by using declarative relationships between the communication messages within the communication protocol. Following the same visual notation of ConDec^{++} we define the following set of message/message relations, as seen in Figure 4-23.

1. **Sequential Messages** (SM): these are the pairs of messages where one of them has to be sent only after the other one is sent. An example from Figure H-1 is when the agent playing the role IR_SupplierAgent has to send the message MSG_RequestFile before the agent playing the role IR_TranslationService can send the MSG_Agree, MSG_Refuse, or MSG_Error. Sequential Message/Message Relation can also be defined with time constraint as shown in Figure 4-13 expresses that message b can be sent only inside the time windows ranging from n to m time units after each time the system participant sends message a. This means sending message b may happen only after a minimum time delay of n and maximum time delay of m time units. MSMAS allows for the branching of this relation, where the source message might have multiple sequential messages relations with more than one message: such a branching constraint is interpreted as disjunctive. The visual notation to express the branching is a dotted line for all connectors originating

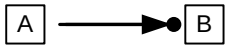
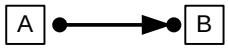
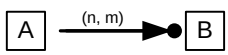
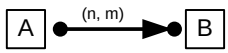
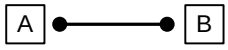
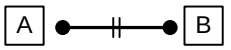
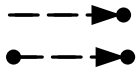
ConDec++ Notation	ConDec++ Visual notation	MSMAS Message/Message Relation
Precedence Relationship: If ***B*** Is performed ***A*** should have been performed before it		Sequential Messages: ***Message B*** has to be sent only after ***Message A*** has been sent
Succession Relationship: every execution of ***A*** should be followed by the execution of ***B*** and each ***B*** should be preceded by ***A***		Joint Messages: ***Message A*** should be followed by ***Message B*** and each time ***Message B*** is sent it must be preceded by ***Message A***
Precedence Relationship with Time Constraint		Sequential Messages: ***Message B*** has to be sent after minimum delay of n and maximum delay of m after ***Message A***
Succession Relationship with Time Constraint		Joint Messages: ***Message A*** should be followed by ***Message B*** after minimum delay of *n* and maximum delay of *m* and each time ***Message B*** is sent it must be preceded by ***Message A***
Coexistence Relationship: If either ***A*** or ***B*** is performed, the other one has to be executed as well.		Coupled Messages: Both ***Message A*** and ***Message B*** has to be sent
Not Coexistence Relationship: If one of ***A*** or ***B*** is performed, the other one can not be executed.		Disjoint Messages: If ***Message A*** has been sent then ***Message B*** can not be sent, and vice versa.
No constraint		Amicable Messages: Any or both of ***Message A*** and ***Message B can be sent*** without any restriction
No constraint		Branching Dotted-line indicate that this relation is part of a set of disjunctive relations, applicable only with Sequential and Joint Relations

Figure 4-23: *DECLARE Notation and its Mapping to MSMAS Message/Message relation concepts*

from the source message to the target messages.

2. **Joint Messages** (JM): these are pairs of messages where both have to be sent, but one after another in a specified order, or neither. An example from Figure H-1 is when the agent playing the role IR_TranslationService sends the message MSG_Agree the agent playing the role IR_SupplierAgent has to send the MSG_SendCustomFile. If MSG_Agree is not sent then MSG_SendCustomFile does not have to be sent. Joint Message/Message Relation can also be defined with a time constraint in the same manner described for the previous Sequential Message/Message Relation. Also branching is allowed with this relation, visually represented as dotted line and expressed as set of disjunctive relations.
3. **Coupled Messages** (CM): these are pairs of messages that are coupled together and both are required to be sent or none, but in either order: once one of them is sent the other one needs to be sent. An example is the requirement in a marketplace for the payments processing bank to notify both the customer and the seller by email about a money transfer transaction involved both of them.
4. **Disjoint Messages** (DM): these are mutually exclusive messages, where only one of

them can be sent at any point in time within an execution instance, and once one of them is sent the other one cannot be sent. An example of this is when there are two options to notify a service either by email or through an API, the protocol might have two messages to support each option but we do not want to replicate the notification process, so once an option is used we forbid the use of other option.

5. **Amicable Messages (**AM**)**: these are the pairs of messages that can be sent at the same time without raising any conflict.

In MSMAS a communication protocol $CommuProt$ is defined as:

$$CommuProt_{msmas} = \langle Msg_{prot}, Msg_{rel}, IR_{prot} \rangle$$

where a communication protocol $CommuProt_{msmas}$ is the set of communication messages Msg_{prot} that are exchanged between the system participants playing the institutional IR_{prot} roles according to the constraints set by the set of relations Msg_{rel} between the pairs of these messages and:

$$Msg_{msmas} = \langle IM \cup OM \cup RM \rangle$$
$$\text{and} \qquad msg_{msmas} = \langle MT, MID, (SID, SIR), (RID, RIR), MSGC, T \rangle$$

where Msg_{msmas} is the set of all messages, and msg_{msmas} is a single message that is defined by its type (MT), its ID (MID), its sender ID and institutional role (SID, SIR), its recipient ID and institutional role (RID, RIR), its content as an object ($MSGC$), and its time stamp (T).

4.5.5 Basic Roles/Activities Models

Basic Roles/Activities Model (BRAM) is an equivalent model to the Basic Business Process Model. It has the same context but different focus. While the basic business process model focuses on which activities are there and how each activity relates to other activities, the BRAM shows, from the institutional role perspective, the ownership of each activity and allows the system designer to specify the system participant/system participant interactions by defining which communication protocol is used to facilitate the activity, or which messages are passed from one system participant to another. Both activity pre-conditions and post conditions are also shown to help the system designer to see the business process, in overview, how the data flows and who owns it during execution. BRAM is similar to the Business Processes Interaction Model (an example model is shown in Figure 2-15 Page 40), with more details, where the inputs and outputs of each activity are expressed as well. Interagent conversations are expressed in this model as one to one only as opposed to multicast. In case of multicast, a series of point-to-point messages can be used to present this requirement.

Each Basic Roles/Activities Model has the set of properties shown in Table 4.13, while each activity has the same properties as detailed in Table 4.10.

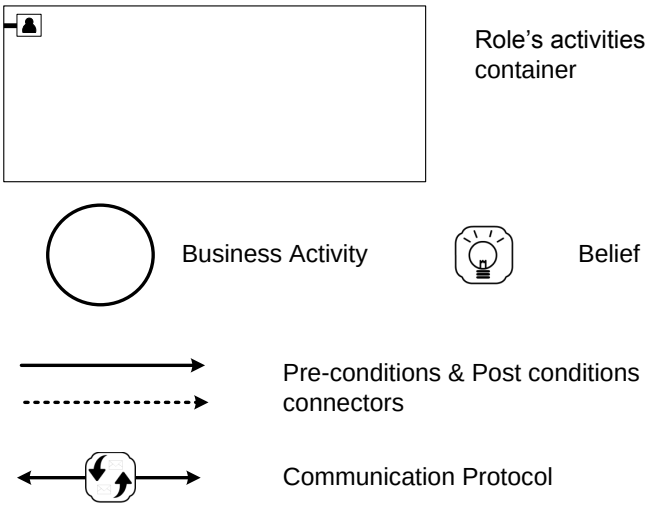


Figure 4-24: *MSMAS: Basic Roles/Activities Model Notations*

Property Name	Data Type	Description
Model Name	String	Unique indicative name of the Basic Business Process it presents.
Description	String	More details to describe this model.
Business Activities	List	A list of all business activities in this basic business process model.
Activity/Activity Relations	List	A list of all activity/activity relations used in this model.
Institutional Roles	List	A list of all institutional roles used in this model.
Communication Protocols and Messages	List	The total number of business activities within this model.
Verified	Boolean	Set to True once the model is verified otherwise it remains false.
Created on Date	Date	In the format ddmmyyyy.
Created on Time	Time	Time formatted as hhmmss.

Table 4.13: *Basic Roles/Activities Model Properties*

Figure 4-24 shows the diagrammatic notations of Basic Roles/Activities Model while, while Figure 4-25 shows an abstract visual model based on previous abstract model shown in Figure 4-20 in Page 116. Notice CP_CommunicationProtocol1 communication protocol between BA_BusinessActivity3 within IR_InstitutionalRole1 and IR_InstitutionalRole3 containers which indicate that this business activity is responsibility of both institutional roles and they use CP_CommunicationProtocol1 protocol to facilitate their interactions. Another example of BRAM is shown in Figure 5-15 Page 181 5-15.

Table 5.20 shows the descriptor of BRAM_PublishSupplierStock Basic Roles/Activities Model:

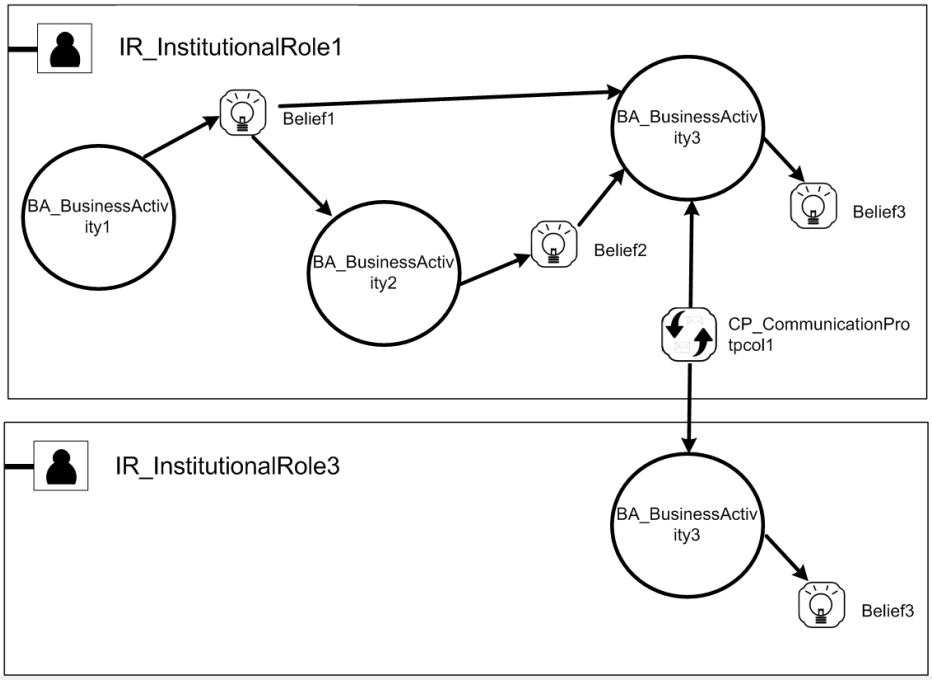


Figure 4-25: *MSMAS: Basic Roles/Activities Model Notations*

By completing all Basic Roles/Activities Models, we reach the end of the detailed design stage and MSMAS second phase. Now we have all models expressed visually and ready for the next phase where we can *verify* the correctness of our models, and check some properties of our designed system to make sure that our design fulfills the system requirements and then start to *implement* our system in the chosen programming language. This is discussed in detail in the following section.

4.6 Verification and Implementation Phase

Verification and validation are common practices in the field of software engineering. In conventional methods verification involves a series of technical and managerial activities that are performed to check that the designed system as a whole or each product of each development phase is consistent with the requirements of that phase and consistent with the overall requirements of the whole system. While validation involves technical and managerial activities that are performed to check that the system operates in a way that matches its specifications.

Developing MASs that are distributed and flexible in their specification and design means their models can be difficult to manage and as a result they are more difficult than others to verify. At the same time, verification reduces the problematic nature of flexibility by vetting model structure. In the first part of this section we highlight which methods, MSMAS uses, for the support of the model verification. The second part is an illustration of mapping MSMAS concepts and constructs to one of current agent framework, namely Jadex.

This phase of MSMAS includes the following activities:

- Review of the system design and verification of the system properties against the requirements and its use cases.
- Generation of the RDF files to be able to transform the system model to any other meta model, or to use in implementing the legacy code.
- Generation of the formal models that enable the verification at run-time.

4.6.1 Verification of MSMAS Models

Verification of the designed models is to answer the question: does the given design correctly capture the business requirements and will the design lead to the building of a system with correct properties? While the validation of implemented system is about checking if the given implementation satisfies the specifications? In this section we limit our presentation to the verification while the validation is discussed in detail in Chapter 6. One of the common practices in computer science is to develop formal theories to specify and reason about computer systems. The agents community, following the tradition, has developed many such formalisms and to reason about multiagent systems, modal logics has been the main approach. The core idea of modal logic is to develop logics that can be used to characterise the mental states of agents as they act and interact [Wooldridge, 2008]. Formal verification includes formal specification, formal model and formal proof either through model checking or other methods.

MSMAS uses both formal modelling and visual modelling and to verify the correctness of system models developed we recommend the system designer to verify the visual models as well as to use formal methods to verify system models. MSMAS models have semi-formal visual notation and a formal representation. Any system designed using the MSMAS tool can be exported to RDF, LTL and CLIMB (Appendix E have brief presentation of MSMAS Designer Tool). Generally, verification of system design is done by checking it is valid according the MSMAS formal model. There are a few principles, which if adopted, we believe makes verification achievable:

- Verification must take place throughout the MAS development life cycle.
- Verification must be done independently and must be documented.
- Verification should be designed to detect the errors as early as possible.
- Verification should be based on formal or semi-formal models.

Montali [Montali, 2010] identified two situations where verification of a system model can be used:

- **Verification of a Single Model**: where the model must be continuously verified to ensure its correctness and consistency, to check if it covers all the requirements, to test if it meets the business objectives, and to assess its compliance with internal policies and internal and external norms, which are not included inside the model itself.
- **Verification of a Composition of Models**: to support the verification of systems that are split into separate components. The process of combining the checking of different

components to effectively realise the correctness of the overall composition.

In this section we list both semi-formal and formal verification features supported by MSMAS to verify system design.

Verification of MSMAS Visual Models

I) Use case models: checked for the following:

- There is not any orphan function.
- There is not any system participant that is not linked to a function.
- All model properties are complete.

II) System Goals Model: checked for the following:

- There is not any orphan goals except the general Goal.
- All composite goals have at least one a subgoal.
- None of the basic goals has subgoal.
- All model properties are complete.

III) Institution Models: checked for the following:

- All role properties are complete.
- All model properties are complete.

IV) Composite Business Process Models: checked for the following:

- All model properties are complete.

V) Basic Business Process Models: checked for the following:

- All activity properties are complete.
- All model properties are complete.

VI) Custom Communication Protocol Models: the model is checked according to these points:

- All messages properties are complete.
- All model properties are complete.
- There is no orphan message, all messages has at least one message/message relation.

VII) Basic Roles/Activities Models: the model is checked according to these points:

- All institutional roles properties are complete.
- All model properties are complete.
- All pairs of activities which are the responsibility of more than one institutional role are linked by a communication protocol.

MSMAS Model as RDF

With the development of semantic web technologies, there has been much research into the use of RDF (Resource Description Framework[8]) and OWL (Web Ontology Language) as representations for models. RDF is a data model with a basic building block of an object-attribute-

[8] www.w3.org/RDF/

value triple, and is called a statement. By taking advantage of XML representation of RDF, it inherits all benefits associated with XML such as machine-readability and validation amongst others. RDF is domain-independent and all terminologies are defined in a schema language called RDF Schema (RDFS). RDF Schema define the vocabulary used in a RDF data model as well as specifying which properties apply to an object, what values it can take, and what relationships exist between objects. The potential benefits of using RDF/OWL to model agents concepts is clear from the research done to combine semantic web technologies with agents and multiagent technologies; such as Laclavik et al. [2006], Dikenelli et al. [2004] and Alberola et al. [2013]. Furthermore, Fornara and Colombetti [2010] uses OWL2DL and SWRL rules to present and monitor norms and obligations.

MSMAS models are fully described as RDF including all system components, their relations and the system norms. The semantics of MSMAS classes and their relations are available as RDF schema. More details on RDF and RDF schema can be found in Appendix C, MSMAS RDF schema can be found in Appendix D while Table 4.14 shows an excerpt from the RDF file that describes an institutional role from our case study.

Having a MSMAS system presented as in RDF allows the system designer to run some checks using the SPARQL query language[9]. Table 4.15 show example SPARQL query that checks that all composite system goals have at least one subgoal. Executing the query against the system model using any suitable query tool, will highlight any problems in the model with regard to this property.

MSMAS Models as LTL

Linear Temporal Logic (LTL) [Clarke et al., 1999] is a form of logic that uses, in addition to classical logical operators, several temporal operators. ConDec[10] has a designer tool called DECLARE[11] that uses ConDec to build visual models and uses LTL as the underlying formal logic language to describe the designed models. Using LTL opens the door to exploit automata generated from LTL expressions for execution of individual services and verification of participating services and whole compositions [Giannakopoulou and Havelund, 2001]. The exported LTL file presenting ConDec models can be imported into the the process mining framework (ProM)[12] for a posteriori verification of properties and service interaction checking. van der Aalst et al. [2005, 2007] have developed an LTL Checker plug-in for ProM framework that allows for conformance checking.

MSMAS allows the system designer to export an LTL file similar to the syntax of LTL files exported from DECLARE tool. Any MSMAS model that uses ConDec relations can be exported as an LTL file which enables the user to utilise the ProM framework for verification.

[9] SPARQL Query Language for RDF: `http://www.w3.org/TR/rdf-sparql-query/`
[10] ConDec notation is based on DecSerFlow langues that was developed to model declarative business workflows
[11] DECLARE: `http://www.win.tue.nl/declare/`
[12] ProM Framework: `http://www.promtools.org/prom6/`

```
<rdf_:InstitutionalRole rdf:about="&rdf_;MSMAS_Ontology_Class10001"
    rdf_:hasId="3030"
    rdf_:hasName="IR_SchedulerManager"
    rdfs:label="IR_SchedulerManager">
   <rdf_:memberOfInstitution
       rdf:resource="&rdf_;MSMAS_Ontology_Class10000"/>
</rdf_:InstitutionalRole>
<rdf_:InstitutionalRole rdf:about="&rdf_;MSMAS_Ontology_Class10002"
    rdf_:hasId="3031"
    rdf_:hasName="IR_SellingChannelListingsManager"
    rdfs:label="IR_SellingChannelListingsManager">
   <rdf_:memberOfInstitution
       rdf:resource="&rdf_;MSMAS_Ontology_Class10000"/>
</rdf_:InstitutionalRole>
<rdf_:InstitutionalRole2RoleRelation
    rdf:about="&rdf_;MSMAS_Ontology_Class10003"
    rdf_:hasDescription="The system participant has to play both Role
        A and Role B"
    rdf_:hasId="3033"
    rdf_:hasName="IR2RR_IR_AdministratorAgent_2_IR_MonitoringAgent"
    rdfs:label="IR2RR_IR_AdministratorAgent_2_IR_MonitoringAgent">
...
   <rdf_:hasInstitutionalRole2RoleRelationType
       rdf:resource="&rdf_;CoupledInstitutionalRoles"/>
   <rdf_:normOfInstitution
       rdf:resource="&rdf_;MSMAS_Ontology_Class10000"/>
   <rdf_:normOfInstitution
       rdf:resource="&rdf_;MSMAS_Ontology_Class34"/>
   <rdf_:institutionalRoleRelationOrigin
       rdf:resource="&rdf_;MSMAS_Ontology_Class40"/>
   <rdf_:institutionalRoleRelationDestination
       rdf:resource="&rdf_;MSMAS_Ontology_Class41"/>
   <rdf_:memberOfInstitutionalRole2RoleRelationsGroup
       rdf:resource="&rdf_;RelationFormulas"/>
</rdf_:InstitutionalRole2RoleRelation>
```

Table 4.14: *An excerpt from the RDF describing an institutional role*

```
SELECT ?goal WHERE {
  ?goal ?o ?p .
  FILTER NOT EXISTS { ?goal this:hasCompositeGoal ?x }
}
```

Table 4.15: *An example SPARQL query to check for any composite goals that has no sub-goals of type CompositeGoal*

```
# version       : 1.0
# date       : 28-07-2013 13:51:50:589
# author     : MSMAS Designer
# Model Name : CSG_ManageSupplierStock SBP
# Model Type : Composite BP Models
# Model Description :
# Number of Roles : 2
# Number of formulae : 0
set ate.EventType;
set ate.Originator;
date ate.Timestamp := "yyyy-MM-dd";
set ate.WorkflowModelElement;
rename ate.EventType as event;
rename ate.Originator as person;
rename ate.Timestamp as time;
rename ate.WorkflowModelElement as activity;
################################################################################
formula not_co-existence_IR_SupplierAgent_IR_StockManagerAgent () :=
{
   <h2>not co-existence</h2>
   <p> Only one of the two tasks <b>IR_SupplierAgent</b> or
       <b>IR_StockManagerAgent</b> can be executed, but not both.</p>
   <p> parameter(s) [A] ->IR_SupplierAgent</p>
   <p> parameter(s) [B] ->IR_StockManagerAgent</p>
   <p> type: mandatory</p>
}
 ! ( ( <> ( ( activity == "IR_SupplierAgent" / event == "complete" ) ) / <>
     ( ( activity == "IR_StockManagerAgent" / event == "complete" ) ) ) )
```

Table 4.16: *An example LTL file describing CSG_ManageSupplierStock model*

Table 4.16 shows an example LTL export presenting CSG_ManageSupplierStock model from our case study.

MSMAS Models as CLIMB

To allow for MSMAS model verification during design time we chose to present them formally as CLIMB models. Our choice of CLIMB in particular was because of the availability of tools that are ready to use and can be integrated and utilised for this purpose. We present in Chapter 6 our mapping of MSMAS models in CLIMB and discuss in details how these models can be verified.

4.6.2 Implementing MSMAS Systems

The implementation is the second and final stage of the last phase on MSMAS methodology, where the designed system is realised as a program written in a programming language and can be deployed in the users chosen agent execution environment. In this section we do not explain in detail how to implement MSMAS systems in all possible target languages, rather we just briefly highlight how some MSMAS concepts can be mapped to the concepts in the

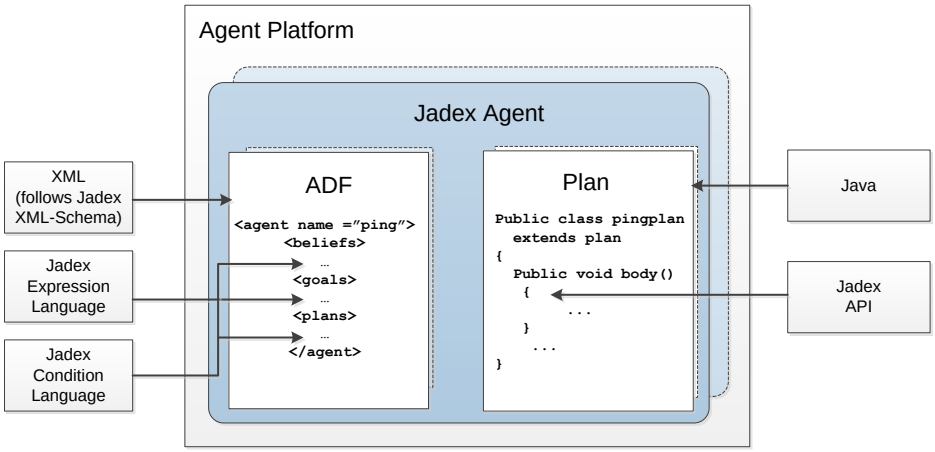


Figure 4-26: *Jadex BDI agent components*

Jadex framework[13]. In particular we explain how the design models can be mapped to Jadex constructs. We then present short excerpts of code. We assume the reader is familiar with Jadex so limit this section to the description of how part of the supplier agent can be implemented.

To develop applications with Jadex, two types of files need to be created: XML agent definition files (ADF) and Java classes for the plan implementations. The ADF is the specification for a class of instantiated agents. For example supplier agents, from the case study are defined by the SupplierAgent.agent.xml file, and use plans implemented, e.g. in the file GetConnection.java. In the rest of this section we describe only the the XML based ADF declaration. Figure 4-26 shows how XML and Java files together define the functionality of a BDI agent and Figure 4-27 shows the top level elements of the ADF file.

ADF File Header

The header of an ADF is shown in Table 4.17, the agent tag specifies that the XML document follows the jadex-2.0.xsd schema definition that can be used to verify that the document is a valid ADF. Each institutional role in MSMAS can be mapped into an agent in Jadex. and the name of the agent type is specified in the name attribute of the agent tag, which should match the file name without suffix (.agent.xml). It is also used as default name for new agent instances, when the ADF is loaded in the starter panel of the Jadex Control Center. The package declaration specifies where the agent first searches for the required classes (e.g., for plans or beliefs) and should correspond to the directory where the XML file is located in. Additionally required packages can be specified using the <imports> tag.

[13] Jadex BDI Agent System: `http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview`, retrieved 15 January 2014

```
<!-- SupplierAgent -->
<agent xmlns="http://jadex.sourceforge.net/jadex-bdi"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://jadex.sourceforge.net/jadex-bdi
                         http://jadex.sourceforge.net/jadex-bdi-2.0.xsd"
  name="SupplierAgent" package="jadex.bdi.OnlineStore.supplierAgent">
  ...
</agent>
```

Table 4.17: *An example ADF file header*

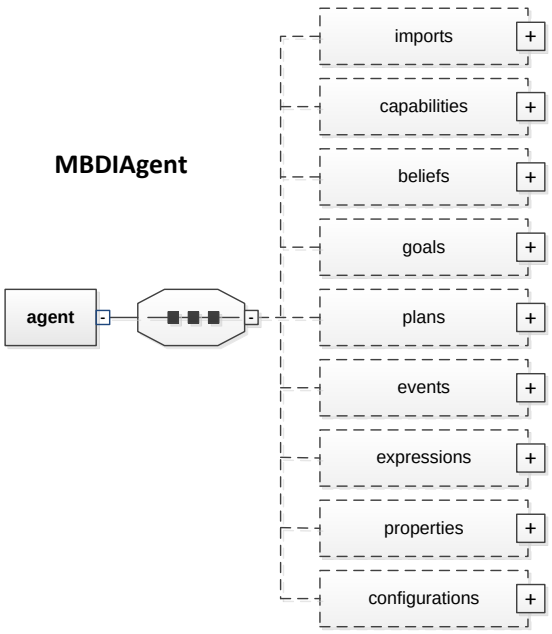


Figure 4-27: *Jadex Top Level ADF Elements*

Defining Beliefs in the ADF

In Jadex the belief base is the container of the known facts of an agent. Beliefs can be any Java object and can be classified into two types: a belief that presents one fact or a set of beliefs which presents a set of facts. For these two types the developer can use the corresponding <belief> or <beliefset> tags and has to provide a name and a class. The name refers to the fact(s) contained in the belief while the class specifies the (super) class of the fact objects that can be stored in the belief. MSMAS beliefs associated with each institutional role can be mapped to and added to Jadex agent beliefbase. Table 4.18 shows how we can present some beliefs of the SupplierAgent

Java implementation of Plans in Jadex allows the user to access the beliefbase using the getBeliefbase() method. The beliefbase provides getBelief(), and getBeliefSet() methods to get the current beliefs and belief sets by name. The contents of a single fact belief are modified using the setFact() method. Setting a fact on a belief will result in overwriting the previous value, if any. Table 4.19 below shows part of the GetConnection plan java file that sets GotFT-

```
  ...
<!-- SupplierAgent Beliefs -->
<beliefs>
     <belief name="GotFTPConnection" class="Boolean">
       <fact>false</fact>
     </belief>
     <belief name="GotMailConnection" class="Boolean">
       <fact>false</fact>
     </belief>
     <belief name="GotCustomFile" class="Boolean">
       <fact>false</fact>
     </belief>
     <belief name="GotSSLF" class="Boolean">
       <fact>false</fact>
     </belief>
   </beliefs>
  ...
</agent>
```

Table 4.18: *An example goals presentations in ADF file*

```
    ...
   <!-- SupplierAgent Beliefs -->


   public void body
   {
    ...
     IBelief gotFTPConnection =
         getBeliefbase().getBelief("GotFTPConnection");
     gotFTPConnection.setFact(new Boolean(true));
    ...
   }
```

Table 4.19: *An example belief presentations in ADF file*

PConnection belief to true.

Defining Goals in ADF

Jadex has four types of goals: Perform Goal, Achieve Goal, Query Goal, and Maintain Goal. We just focus our example on Achieve Goal which is used to reach some desired world state. Each business activity in a MSMAS system associated with a subgoal of MSMAS basic system goal is mapped to a Jadex achieve goal. This type of goal has <targetcondition> which are the postconditions of the business activity. The target condition is used to specify in which cases a goal can be considered achieved. These are the MSMAS business activity postconditions. The example in Table 4.20 shows how we can specify the BA_GetConnection subgoal of BSG_PublishSupplierStock:

```
<achievegoal name="BA_GetConnection">
  <targetcondition>
    $beliefbase.GotFTPConnection.equals(true)
  </targetcondition>
  <targetcondition>
    $beliefbase.GotMailConnection.equals(true)
  </targetcondition>
</achievegoal>
```

Table 4.20: *An example goals definitions in ADF file*

```
<plans>
 <plan name="getConnection">
   <body impl="getConnectionPlan"/>
   <trigger>
     <condition>\$beliefbase.updateTime.equals(true)</condition>
   </trigger>
 </plan>
</plans>
```

Table 4.21: *An example plan presentations in ADF file*

Defining Plans in the ADF

Plans in Jadex present the agent's means to act in its environment and they are normally seen as responses to occurring events or goals. The selection of plans is done automatically by the system and presents a main aspect of the BDI infrastructure. In Jadex, plans consist of two parts: a plan head and a corresponding plan body. The plan head is declared in ADF, whereas the plan body is realised as a concrete Java class. The example in Table 4.21 below shows how we can specify the BA_GetConnection system plan head in the ADF, that is triggered based on a condition:

As mentioned in the beginning of this section, we do not aim to present a complete implementation example of the MSMAS designed system. Instead our intention was to demonstrate how to implement MSMAS agents and some of the MSMAS concepts using Jadex. Part of the planned future work is to export from MSMAS directly to a MAS platform. Jadex is one of the top candidates to be supported.

4.7 Self-Managing Modelling Approaches

We listed in Chapter 2 Section 2.8 Page 54 various approaches to model a software system with self-management properties. To illustrate how we can follow one or more of these approaches, let us consider the following scenario based on our case study. In the suppliers stock management system, it is assumed that some external agents may join the system, these agents might be developed by the suppliers they present or by a third party such as a contractor software development company. As a result, there is no guarantee that these agents will comply with

the system rules all the time. So the question now is, how the system can monitor the supplier agents' compliance with system norms and how to handle a violation when it is detected. We discuss briefly in the following sections how to address this requirement using various modelling approaches, then we present a set of models with the required changes to introduce a degree of self-management to our system.

4.7.1 Dynamic Components Approach

The key elements in modelling dynamic components-based systems are (i) the definition of the component properties, (ii)identifying the component dependencies, (iii) and having a mechanism for resolving the dependencies. By definition, multiagent systems are an implementation of these principles and self-management follows from different initial assumptions, such as the ability of agents to communicate, coordinate and, cooperate in a social manner. Modelling the dynamic components in MSMAS is achieved through the definition of the goal-based business processes and activities. The system designer may think of each component as a subsystem represented by one composite system goal and served by one institution that contains a collection of institutional roles which are responsible for the achievement of all goals and the execution of all functions/activities within this subsystem. Both of the system goals model and the composite business process models can then be used to define and refine the relations between all modelled subsystems. The components properties are then described by means of system goals and goal/goal norms, and the model could contain system components that are able to search/plan the agents activities through discovery or directory service. In the BBP_PublishSupplierStock Basic Roles/Activities Model example shown in Figure 5-15 we showed how we can create multiple business activities with the same postcondition, following this example the system could be modelled with multiple business processes that lead to the same system state.

4.7.2 Central Control Approach

Modelling a system with either an internal or external control mechanism requires the system to be self-aware and have the reasoning capability required to interpret its behaviour and adopt changes. The use of the internal control mechanism is more suitable for a focused local view of adaptation such as exception handling. In this approach the control mechanism is normally mixed up with other system components where the adaptive components perceive their changes, other components' changes or the changes taking place in their environment, then they use their reasoning ability to verify these states and apply some actions to rectify their behaviour or affect the environment or the observed components. Figure 4-28 shows an abstract design of a system with internal control based self-managing mechanism. Please notice that this mechanism may be hard to maintain or scale up.

External control mechanism, in contrary, depends modelling the system as two sub-systems,

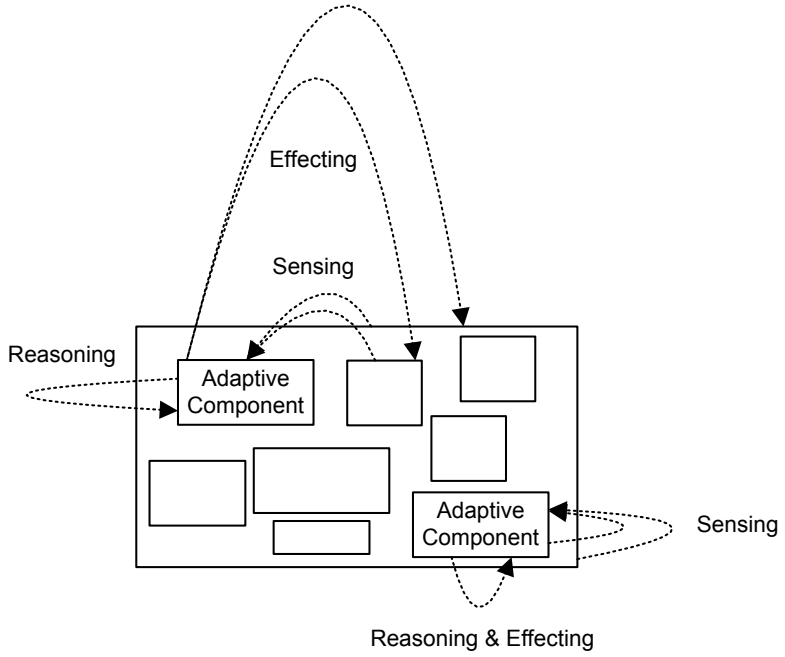


Figure 4-28: *An Abstract Model of a Self-managed System with an Internal Control Mechanism*

the first is the functional system that operates to conduct the business goals. The second subsystem is an adaptive engine that is able to sense the global behaviour of first subsystem or only senses a number of components in that system. The adaptive engine may have also the ability to affect the functional subsystem such as isolating/replacing a faulty component. Figure 4-29 shows an abstract design of a system with an external control based self-managing mechanism. This mechanism may be suitable for situations where the main focus is the global behaviour of a system and works well with both model driven and dynamic component approaches. It is also easier to scale up and reuse across multiple systems.

In MSMAS both control mechanisms can be modelled in many ways. One way is at the institution level, where for the internal control the system designer can create a single institution with one or more institutional roles with authority over the remaining institutional roles using the Child/Parent Roles Norm. The parent role can then delegate tasks to the child role, and the child role can report back the progress. The parent role can then reason and adjust the behaviour through delegating other tasks according to a policy or logical verification. To model the external control at the institution level, the system designer can create two institutions one for the functional subsystem and the other for the adaptive one. The system participants in the adaptive institution can then be responsible for all monitoring or checking the logs of the functional subsystem as well as conducting the control activities such as changing the functional system institutional role/role norms in the way that can isolate the faulty system participants playing particular institutional roles.

Another way of modelling the control mechanism is through the business activities, by

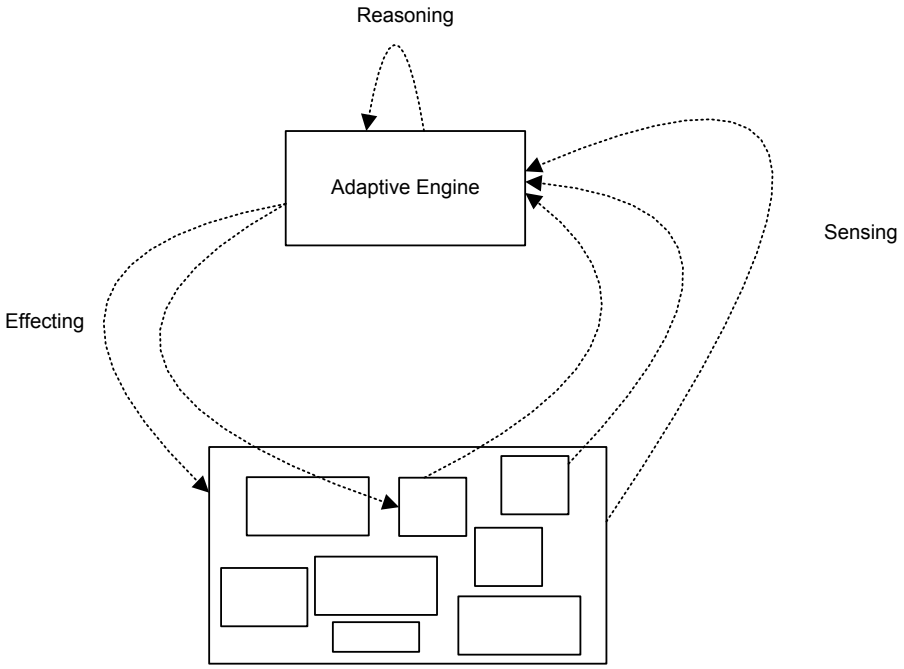


Figure 4-29: *An Abstract Model of a Self-managed System with an External Control Mechanism*

introducing a set of activities including logging the system participants interactions, and reasoning activities for the checking the system logs for anomalies and a number of corrective activities to be executed to rectify the system state. These control related activities can then be assigned and used by each self-adaptive component in case of internal control, or by the adaptive engine subsystem, in case of external control approach. Section 5.2 Page 169 includes some example models based on our case study that illustrate how we can implement the control mechanism as well as other self-management mechanisms.

4.7.3 Feedback Loops Approach

As explained in Chapter 2 Section 2.8 most, if not all self-managing autonomous systems have a closed feedback loop. The main required components to create a closed feedback loop system are: i) A component that monitors the state of a (sub-)system. ii) A component that calculates a corrective action. iii) A component that applies the corrective action to the (sub-)system. To model this in MSMAS we can create invitational roles to be responsible for each set of activities corresponding to each component. For example, we create an admin agent that to be responsible for approving new agents to join and suspending violating agents. The second role is a monitoring agent role to be played by one or more agents and be responsible for reeving reports or checking the system logs for anomalies. And the third role is a planning agent that reason about the current state and find appropriate plan to rectify the situation. The plan then can be executed by other system participants or enforced by the admin agent. The examples in Section 5.2 Page 169 show how a feedback loop can be modelled in MSMAS

while the violation detection and the application of a corrective action is presented in Use Case II (Suspend Non-compliant Supplier Agent).

4.7.4 Model Driven Approach

Modelling a self-managed system according to the model driven approach, requires the availability of run-time monitoring and machine-readable models that describe the system components. MSMAS offers a full representation of the system components as RDF, and a formal representation of the system norms as CLIMB. Both models can be used with the right tools to satisfy this approach. For example the code generation of part or all of the of system components is possible, the availability of logic reasoners makes it possible to monitor the system execution at run-time, then stopping the faulty component once a violation has been discovered and a replacement of a newly generated component can be done. We propose another perspective on model driven self-management systems that utilises the formal model of system norms to achieve a degree of dynamics of norms. By creating a number of system administrator institutional roles who can be responsible for issuing and publishing obligations as system norms and publishing them publicly. All system participants can then use these rules as guidelines for their expected behaviour. Agents that play the admin roles can also monitor and react to any violations by using a monitoring mechanism to detect the violating agents and then enforce the system norms through the application of sanctions. Achieving a full dynamic normative system is also possible by allowing the norms issuing agents to observe the emergent behaviour[14] and adopt or issue new norms that reflect these emergent norms. Figure 4-30 shows an abstract model of our proposed architecture for the implementation dynamic norm self-managing system that combines model driven approach with feedback loop and external control approaches. In this model, there is a normative engine modelled as an external control component, it uses a repository of constitutive norms that define the mechanism of issuing, modifying and abolishing the regulative norms. The normative engine with the monitoring mechanism form a closed feedback loop, where violations and compliance with the active norms are detected then enforcing/rewarding actions are taken accordingly. Finally, constitutive and regulative norms are presented as formal models and accessible by the normative engine and other system components.

4.8 Guidelines for Using MSMAS

System modelers are encouraged to think of MSMAS as a set of guidelines, not rules to be followed strictly. Although a few steps of MSMAS require the modeler to do things in specific way/order, we are very clear and insistent on keeping it balanced to maintain flexility, to fit many scenarios and a wide range of application domains.

[14] At the implementation level this can be done using Inductive Reasoning techniques such as likelihood and Bayesian estimation

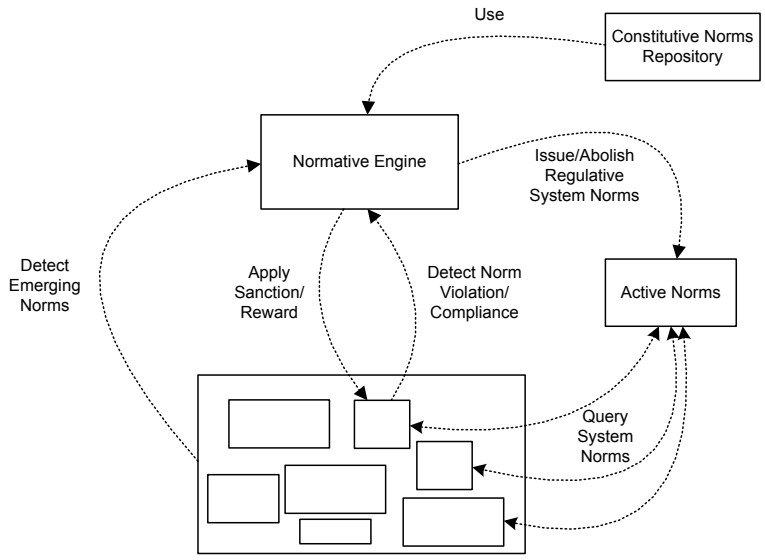


Figure 4-30: *An Abstract Model of a Self-managed System with dynamic norm mechanism*

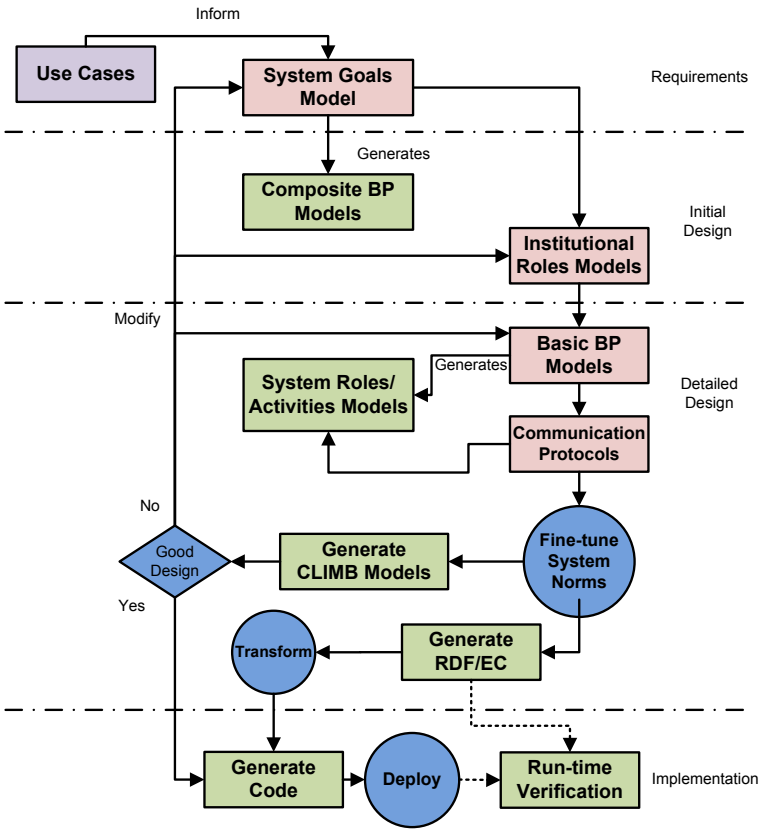


Figure 4-31: *Suggested Activity Sequence for System Modelling, Verification and Implementation Using MSMAS Methodology*

Although, we believe that MSMAS is suitable for a wide range of domains and users, we do not present it as the single best methodology to be used for all situations. Similarly to other

MAS methodologies, MSMAS will evolve over time in solving modelling issues as they appear and its tool (Appendix E) will evolve as well giving more freedom to the system designers especially when it comes to allowing for different start points. In this section we provide some guidelines on how to utilise MSMAS to create MAS Models. We, however, encourage MSMAS users to apply common sense and logical judgement and feel free to manoeuvre as required based on their skills and application domain.

1. **Requirements Gathering and Specification**

   Business Requirements and System Specification are always the first phase in the full cycle of software development. It often begins with a set of meetings with stakeholders and initial documents that include a few paragraphs of generic description. By the end of this phase the system designer should have a clear and precise understanding of the system to be developed. The greater the efforts put towards this phase, we believe, the less the chance of encountering problems at later stages.

   Figure 4-31 shows a suggested sequence for the modelling process. We suggest that the system designer starts sketching many scenarios that cover the most important business processes the system has, using the use case models. Thinking through these scenarios will help in identifying the main system participants and the building of a common knowledge of sequence of events, as well as identifying ambiguous areas of the system and revealing the well understood features from the meetings and initial documents he might have received.

   Use case models do not enforce any design decision on the system as yet, they are however useful in helping the system designer in identifying the system goals and their sub-goals. The development of these use cases serves as a visual reference for the system designer and other stakeholders of the system processes and the various responsibilities of both internal and external system participants.

   After collecting all business requirements at high level, and with the help of the use cases, the designer has to critically assess if the system has to be agent oriented or not. The designer may consider a few parts of the system to be suitable candidates for designing as agent-oriented, while the remaining parts remain non-agent software. It is largely acknowledged that not all software system components are best modelled and designed as agents. For example all static components are better as either data stores or services. We should use agents where they offer a benefit and are expected to play proactive and interactive roles. It is worth remembering that agents are autonomous, goal-directed, multi-task-oriented, behave dynamically and proactively.

   The next step is to build the system goals model, which is a tree of composite and basic goals and sub-goals with the relation indicator that defines sub-goals of each system goal. Goals are the natural constructs to use when building the system specification: they are central to the system participants and the core reason for each business process. System goals model determines the structure of other models in MSMAS, namely business

process models and the system roles models.

The system goals can usually be extracted from the requirements, the designer can identify the goals and structure them through goal break-down process. The purpose of identifying the system goals is to derive the overall system goal and its subgoals. Once the goals have been identified, structuring them is done by defining the goal/subgoal relationships. Each goal G which represents an end state is defined through its decomposition into non-cyclic subgoals. For each goal G there are multiple ordered lists of leaf nodes (basic goals) G_L that lead to the achievement of that goal. These lists can be understood as a disjunction of conjunctive predicates. The order of the goals is defined through either setting one-sub goal as a trigger of another sub-goal or through defining one sub-goal to proceed another sub-goal. At this phase we do not define if these sets of sub-goals are contributing to the achievement of their super goal totally or partially. The AND-refinement or OR-refinement is not done in the system goals model but in the next design phase at the composite business process level.

Yu and Mylopoulos [1994] have provided a suitable method to reason about business processes and conduct and define the system goals and their sub-goals. Once the designer have completed the system goals model he/she can move to the initial design stage.

2. **System Design: Initial Design** The first step in the initial design stage is to refine the relationships over the system goals, by setting the system goals norms. This can be done by visiting each composite business process model and defining any number of needed goal/goal relations which can allow the designer to set the AND/OR relations and others explained in more details in section 4.4.3. The next step is to specify the organisational structure of the system. In MSMAS there must be at least one institution and to utilise the features of self-management the designer needs to include in his/her design an institutional role that governs that institution and be responsible for the observation of the system norms and the enforcement of these norms. Some systems might require multi institutions to distinguish between different business functions or to implement a group-based privilege system, where each group (institution) enjoys different access level or different set of privileges. The system governor must in this case be a member of all institutions implementing a distributed governed system. In this case each institution governor must be member of the governors group to allow for communications, and information sharing, between all governors.

   In each Institutional Roles Model, you can specify any number of role/role relations (system roles norms) that would allow for controlling the behaviour of the system participants that play a particular role in relation to other roles. By the end of this stage the designer is expected to have a complete set of Institutional Roles Models and refined set of Composite Business Processes Models, then he/she can move to the detailed design stage.

3. **System Design: Detailed Design and Verification**

This stage starts by adding the fine details of the system Basic Business Processes. In each model the designer can specify any number of business activities. The business activity must have post-condition which is any number of beliefs, may have any number of pre-conditions (system beliefs), and is executed by any number of system participants playing specific institutional roles. If the activity is executed by more than one system participant then defining which communication protocol is required in the next step: the refinement of System Roles/Activities models. Once the designer has created the needed business activities he/she can specify any system norms between these activities. These system norms are activity/activity relations that can be used to control the sequence of execution in a declarative manner. For more details on applicable activity/activity relations see Appendix F.

There is no particular order the designer need follow to complete the required business activities for each basic business process model, the creation of communication protocols and the refinement of system norms on the business activity. A designer might choose to create all communication protocols before assigning any of them to each applicable pair of activities in the system roles models, while another designer might create then assign one by one.

It is very rare for a system designer to get everything right the first time, so it is expected to go through a number of iterations to each model and change his/her previous design decisions. Some models might need the whole process (requirements specification, initial design, detailed design, and implementation) to be iterated with change of emphasis during each iterations. Although we present MSMAS in a sequential manner, we encourage the system designer to follow any other sequence that he/she might see as more suited for his/her application or his business domain. The tool (Appendix E) we have developed does not currently support consistency checking, but inconsistencies can be introduced when the user modifies one of the designs, in a way that requires changes in other models. Our future plan includes implementing automated crosschecking functionality to help the system designer avoid this issue. For now, the system designer has to perform such cross-checking manually.

The completion of the detailed design means the designer has created all needed business activities in each Basic BP model, created all communication protocols, assigned a communication protocol to each pair of activities in the system roles/activities models where applicable, and fine-tuned the system norms by specifying the required activity/activity relation. The designer can then verify his/her system model by generating the CLIMB models and using the $\mathcal{S}$CIFF proof procedure to check the model properties or to discover inconsistency, he/she can also use g-SCIFF to query some system specific properties; for more details on how to verify the design, see Chapter 6. If the designer is happy with his/her design, then he/she can generate the RDF model where it can be used to transform the design to another modelling language or used as a base to implement

the system in legacy code. Our future work plan includes implementing an export function that allows for the generation of this system into a skeleton code in a chosen target programming language such as Jadex, JASON ...etc.

4. **Deployment and Monitoring** At the moment we do not recommend any specific programming language. Our future plan for MSMAS supporting tool includes the implementation of automatic generation of legacy code. The system developers will need however to complete the details of the code and testing. One feature that distinguishes MSMAS from other methodologies is online monitoring. The system developer can export the formal models of his system and by generating traces of the execution events and the use of an Event Calculus based monitoring tool, he/she can monitor the execution of the system and allow for self-management if the design included correction functions that are triggered when there is any type of violation detected. For full details on the syntax of MSMAS events and how Event Calculus based monitoring can be used see Chapter 6, Section 6.7.

4.9 Intra-System Norms Relations and Discussion

MSMAS has well-defined relations between System Goals, Business Activities, Institutional Roles and Communication Messages. According to the MSMAS metamodel – presented in Chapter 3 – a system participant playing an institutional role becomes responsible for the execution of business activities associated with this role to achieve their associated system goals. System participants may communicate with other system participants while executing some business activities. This relation between the system components that may be constrained by system norms leads to the following questions:

1. How does a violation of a system norm affect the state of other system norms, either from the same system norm group or from a different group?
2. Do system norms of particular type have authority over other system norm types?

MSMAS does not include an identification of these intra-system norms relations. Instead we leave them to the implementation because we believe that each application domain has different use cases that will dictate how these intra-relations are set and managed. MSMAS aims to remain a generic MAS development methodology, supporting a wide range of application domains, hence MSMAS does not specify particular norm-norm relation. However, we note that this is an area that needs attention and is a matter for future work informed by experience.

4.10 Chapter Summary

In this chapter, we have detailed our proposed MAS development methodology MSMAS. Our study of other MAS methodologies has influenced our design decisions of MSMAS to avoid the issues found in these methodologies and to mirror their good features, steps, and concepts.

The MSMAS methodology has three phases covering the full life cycle of developing MAS. MSMAS has seven visual models in total supported by a number of descriptors of the various system components. The first phase of MSMAS is focused on requirement gathering. By the end of this phase the system creates a number of use cases models and the system goals model. The second phase deals with the design, where in its first stage it focuses on building the system organisational structure and fine tuning the relations between system goals. The second stage focuses on the detailed design, of business activities and mapping them to institutional roles as well as specifying which communication roles to be used. The last phase covers the implementation where the design can be exported to various formats and the system constructs can be mapped to actual code. Using the supporting tool, the system modeller has to create only four models while the remaining three models are automatically generated, and used for specifying more details about the system norms. We presented briefly a number of approaches for the implementation of self-management features. We have also proposed an abstract model for modelling the normative MAS that uses external control through a normative engine and feedback loop mechanism for monitoring and enforcement. Following that, we have presented a set of guidelines for using MSMAS supported by an activity sequence for system modelling. MSMAS offers various formal representations of its models such as RDF, LTL, and CLIMB. The availability of these formal models makes it possible to reason about MSMAS models and checking their correctness.

MSMAS distinguishes itself from many other methodologies by its coverage of the full development life cycle and the inclusion of an implementation phase. Finally we have illustrated how MSMAS model constructs and concepts can be mapped to legacy code in Jadex to demonstrate MSMAS coverage from design through to to code.

CHAPTER 5

CASE STUDY AND SELF-MANAGING MODELLING EXAMPLES

In the previous chapter, we presented in detail the three phases of MSMAS methodology as well as its visual models and their notations. In this chapter, we complement our presentation with the case study to illustrate MSMAS design process and to highlight its features and abilities to model specific use cases based on real world application. In Section 5.2 we expand our case study examples to illustrate how to use the self-management modelling approaches presented earlier in Chapter 4 Section 4.7.

5.1 Virtual Stock Control and Offers Management System

Case Study	**Virtual Stock Control and Offers Management System**
Summary	A system that supports the selling of goods online across multiple electronic marketplaces and manages multiple availability data feeds from multiple suppliers

Table 5.1: *Use Case Summary: Virtual Stock Control and Offers Management System*

Our case study includes the following:

1. **A description** of the problem that explains what is the system intended to achieve? Any system is designed to solve a specific problem(s). The description of a system architecture is complete with the inclusion of the type of problem it solves.
2. **A philosophy**: What is the reasoning behind choosing this method of MAS paradigm to solve this problem? What are the central ideas in the architecture?
3. **Modelling Examples**: How to model each system component. We follow MSMAS steps to create a number of visual models and their descriptors to illustrate how each specific requirements is modelled.

5.1.1 Case Study Description

Online retailing has become a common activity for both traditional retailers as well as the early adopters of e-commerce, through marketplaces selling or strictly *only* e-retailers. Nowadays, an increased number of companies advertise their goods not only through their own direct-to-consumer web sites, but also through third party mediators know as marketplaces which process the transaction between the consumer and the selling company.

Selling in these marketplaces creates a situation whereby such companies have to compete with a large number of e-retailers at different levels [Elakehal and Padget, 2012a]. Consequently each company strives to increase its chances through offering a large number of products at competitive prices and comparable service levels. To create and maintain a large catalogue of products offered there is a need to establish, quickly, a complex network of peer to peer (P2P) relationships with a large number of suppliers and manufacturers of very different sizes who each utilise a variety of different trading and data interchange standards [Elakehal and Padget, 2008]. Increasing the product selection to include as many products as possible, leads to a long-tail business model where the company offers and sells a few of the huge variety of offerings with niche interest. This business model is however not possible with traditional inventory acquisition and management methods due to logistical as much as business reasons, such as limitation of space, and cash flows to cover stocking even one of each product, additionally because of the uncertainty around the future demand, any forecasting system could fail in predicting how much units to stock per each product type. To overcome these issues, the company might choose to hold a limited number of products, in stock while fulfilling the remaining customer orders either directly from the suppliers who work as *drop shippers* or by sourcing the products on demand from the suppliers once the company receives the customer order. In this situation an e-retailer has evolved to become more of a facilitator of producer-customer relations, where it needs to maintain a virtual warehouse with virtual stock rather than the traditional physical warehouse. So in our use case the company does maintain large number of relations with suppliers of various products, and use this data to produce large number of offerings to its customers over various selling channels at various prices based on competition level in each channel. The problem then is characterised by:

(i) Providing a continuously available system.
(ii) Consolidate a variety of data formats provided by the different service providers.
(iii) Provide an accurate stock level and offer prices to various external systems.
(iv) Monitoring the data for anomalies and execution times to preserve accuracy.
(v) Provide generic system that able to scale up without major changes or outage periods.

The technical challenge is how to handle:

- Large catalogue that contains millions of products.
- Different data delivery frequencies.
- Various data formats in known standard forms as well as custom forms.
- Inconstant data quality, where errors do come unexpectedly.

- Different data delivery methods, some data is pushed and other are to be pulled over various protocols.
- Multiple data sources where a number of items are available from multiple suppliers with service levels in terms of delivery lead times plus fulfilment rates and different commercial terms in terms of discount values.

The problem presented above is a very good example of a dynamic application domains where MAS can provide a good solution. The problem is complex enough to justify the use of MAS paradigm rather than other modelling and programming techniques. The main features are the dynamic nature of data, and explicit need for scalability where the system needs to grow over time. The table below has more details on why we chose this problem in particular and why MAS is good choice to address this problem.

The problem presented above is a very good example of a dynamic application domains where MAS can provide a good solution. The problem is complex enough to justify the use of MAS paradigm rather than other modelling and programming techniques. The main features are the dynamic nature of data, and explicit need for scalability where the system needs to grow over time. The table below has more details on why we chose this problem in particular and why MAS is good choice to address this problem.

5.1.2 Case Study Philosophy

We have chosen this particular problem to demonstrate the modelling process using MSMAS methodology, motivated by the following reasons:

(i) Virtual stock control is becoming a common problem for almost all companies trading online especially with a large selection of products.

(ii) This problem is a good example for building distributed and scalable systems, where the system expand over time to respond to new requirements and increasing transaction volume.

(iii) The problem is neither too easy in a way that traditional software paradigms fit best as a solution, nor too complex that we will need to expand the modelling over very large space and it becomes too large to comprehend specially for readers whose primary interest is to understand how MSMAS can be used to model MAS systems.

(iv) This particular problem requires maintaining an open-architecture approach when modelling that part that allows external service provides to submit data and maintain their availability information, at the same time the central catalogue and its maintenance requires closed-architecture approach to limit access to data and other business logic functions to only internal parties. That is a good use case for us to illustrate how to utilise the institution structure.

(v) This problem requires methods for errors handling at both the data level and the execution activities, which requires defining monitoring and replanning techniques to achieve a degree of self-management.

5.1.3 Case Study Models

In this section and the following subsections, we present a number of visual models created using MSMAS techniques. We follow MSMAS modelling steps as detailed in Chapter 4. Under each step, we present at least one visual model to demonstrate the final product of that step. We start the System Requirements Phase by creating the required use case models that describe the important scenarios in our system, then we build the system goals model.

Use Cases Models

Let us consider the following scenario, a supplier agent submits its updates to a central stock manager the frequency of updates are dependent on the availability of a new update file. So whenever the supplier agent gets its update file, which could be either in custom or standard format, it submits it to the stock manager. Though the stock manager accept only standard format files. As a result the supplier agent is expected to validate the file before proceeding to next step. If the update file was in a supplier-custom format then the supplier agent has to translate it to the system standard format (SSLF) by submitting the file to the translation service which returns a standard format file. The supplier agent can then submit its standard update file to the central stock manager agent.

In the following context we create our use case model to demonstrate how these models can be used to clarify the system requirements and be a first step towards the identification of the system goals and possible system roles. Figure 5-1 shows the visual use case model while Table 5.2 shows the descriptor of the use case model with all details. In the visual model, the connectors between the system participants and the use case functions, in the use case graphical model, indicate that the system participant is taking part in that function, and the arrows between functions show the possible execution sequence that matches the normal flow. Notice that during the creation of the use case model, the system modeller can start identifying the type of the system participant (Agent - Service - Actor - Environment), if the system modeller is not sure then the the generic system participant icon can be used. When naming the system participant, it is preferred to name them by the role they might play in that scenario. Following this tip helps in identifying the roles needed for the organisation structure.

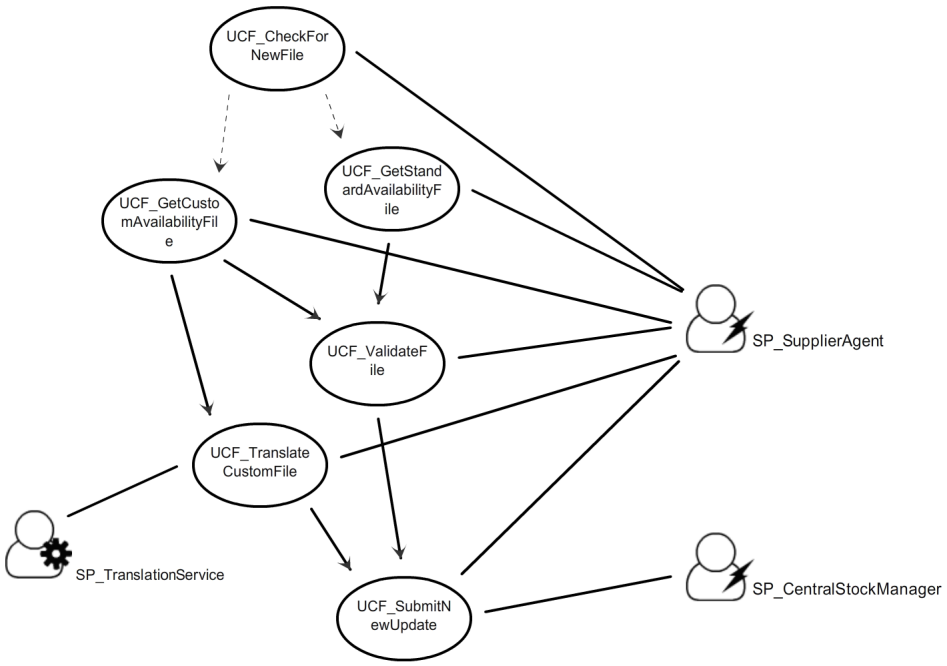


Figure 5-1: *Use Case Model I: Publish Supplier Stock Levels' Update*

Case Study	**Virtual Stock Control and Offers Management System**
Use Case ID	**02**
Use Case Name	**Use Case I: Publish Supplier Stock Level Update to the Central Stock**
Description	Each agent presenting a particular supplier should be able to submit an update file that contains the latest quantity count per each item held in stock. The update file could be a full update or a delta update, the former contains a full list of all items held in stock and their available-to-order quantity, while the later contains only a list of those items that changed in quantity since last update. The process starts whenever the Supplier Agent (SA) senses that a new update file is available and it believes that it is the right time to send its stock updates. After the new update file is downloaded the supplier agent submits it directly to the Central Stock Manager Agent (CSMA). The file needs to be in the Standard Stock Level Format (SSLF) to be accepted by the CSMA. SAs that have their updates in a custom format, may use a translation service to transform their custom-format-file to SSLF.
Goals	1. Manage Supplier Stock (a) Get delta updates (b) Get full updates (c) Get deletes updates (Out of stock) (d) Publish supplier stock updates 2. Translate supplier custom files into SSLF and vice versa

Continued on next page

Table 5.2 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Preconditions	Stock update time and new update file available
System Participants	In this use case there are four different system participants: 1. **SP_SupplierAgent:** an agent that presents an actual supplier in the current system and is responsible for providing accurate information about the supplier physical stock levels and cost. 2. **SP_TranslationService:** one of the system helper agents that are available to facilitate some actions and allow for making the system open for those agents that might not be well equipped with all capabilities. This system participant is specialised in transforming supplier-custom format file into the standard format SSLF and vice versa. 3. **SP_CentralStockManager:** the system participant that is responsible for managing the central stock where the consolidated quantities for each item available from all suppliers that can supply these items are stored. It is responsible for keeping this virtual stock as accurate and close to real world physical stock to avoid overselling some items and losing synchronisation.
System Functions	As shown in Figure 5-1 we have created the following six functions: 1. **UCF_GetCustomAvailabilityFile:** a business process to be used by supplier agents that can provide their specific custom format where the supplier can get its custom stock levels file from either its FTP server or from a mail server. 2. **UCF_CheckForNewFile:** a business function that allows the supplier agent to check if there is new update file available to download. 3. **UCF_GetStandardFile:** a business process to be used by supplier agents that can provide the standard stock level format (SSLF) file where the supplier can get its SSLF file from either its FTP server or from a mail server. 4. **UCF_ValidateFile:** do a check on the file naming, size, and any other variables that can be checked to assess the integrity of the file contents. 5. **UCF_SubmitNewUpdate:** a generic function where the supplier agents can submit their updates to the central stock manager. Only accepted file format is SSLF. 6. **UCF_TranslateCustomFile:** a function where the supplier agents can translate their custom format file into the standard file format SSLF.

Continued on next page

Table 5.2 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Normal Flow	• SA downloads the updates file either from the mail server or from FTP server. • SA checks its file to ensure it is complete and valid. • SA requests file translation to SSLF in case it is a supplier-custom-format file. • SA requests to submit its update file to central stock manager. • CSMA confirms the receipt of the update file.
Exceptions	• SA is not able to connect to FTP server or mail server. • File fails soft consistency checks. • No response from translation service. • No response from CSMA.
Notes	None

Table 5.2: *Use Case I: Publish Supplier Stock Levels' Update to the Central Stock*

System Goals Model

Creating the system goals model depends heavily on a system description that normally contains implicit and explicit indications of system goals and the use case models that were created in the previous step. In these use case models, we have identified an initial list of system goals, these should form the base of the tree and the current task would be around re-assessing these goals, and grouping some of them or extending others. For example, consider the Use Case 5.2 in which the following goals are identified:

1. Manage Supplier Stock
   (a) Get delta updates
   (b) Get full updates
   (c) Get deletion updates (Out of stock)
   (d) Publish supplier stock updates
2. Translate supplier custom files into SSLF and vice versa

The designer should now consider segregating these goals into three types: some goals will end up as composite goals, others as basic goals, and the rest will be activities within a basic goal. If the goal can be broken down into a number of sub-goals then it is either a composite goal or a basic goal. If it is too generic and its scope is too wide, then it is definitely a composite goal, while a basic goal should present only one function.

We have created the system goals model, shown in Figure 5-2, based on the identified goals in our use case model plus few other goals that were needed. The system goals model

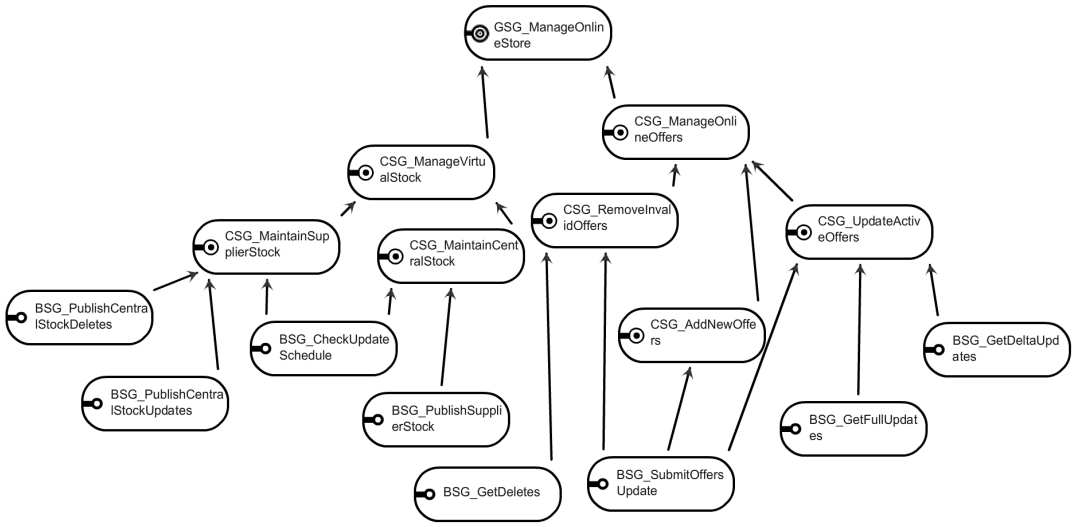


Figure 5-2: *System Goals Model: Virtual Stock Control and Offers Management System*

contains one general goal, seven composite system goals and eight basic system goals, that cover all identified three core functions of the system. Table 5.4 shows the properties of CSG_ManageOnlineOffers system goal, as an example of a system goal descriptor.

Case Study	**Virtual Stock Control and Offers Management System**
Model Name	**System Goals Model**
Description	From the initial system description and previous use case models, the system goals tree should cover three areas to meet the business requirements: 1. Manage the virtual stock availability by allowing the the suppliers to feed their availability information to the system. 2. Manage the selling offers, where the system can send its virtual stock availability to various selling channels.

Continued on next page

Table 5.3 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
System Goals	GSG_ManageOnlineStore 1. CSG_ManageVirtualStock (a) CSG_MaintainCentralStock i. BSG_PublishCentralStockDeletes ii. BSG_PublishCentralStockUpdates (b) CSG_MaintainSupplierStock i. BSG_PublishSupplierStock ii. BSG_SubmitEventsReport 2. CSG_ManageOnlineOffers (a) CSG_RemoveInvalidOffers i. BSG_GetDeletes ii. BSG_SubmitOffersUpdate (b) CSG_AddNewOffers i. BSG_GetDeltaUpdates ii. BSG_SubmitOffersUpdate (c) CSG_UpdateActiveOffers i. BSG_GetDeltaUpdates ii. BSG_GetFullUpdates iii. BSG_SubmitOffersUpdate
Goals Count	• General Goal: 1 • Composite Goal: 7 • Basic Goal: 8
Created On Date	03112013
Created On Time	132015

Table 5.3: *MSMAS System Goals Model: Virtual Stock Control and Offers Management System*

Note that for each system goal in the system goal model the system goal descriptor must be completed. We present in Table 5.4 the properties of CSG_ManageOnlineOffers system goal, as an example.

Case Study	**Virtual Stock Control and Offers Management System**
Name	**System Goal Descriptor**
Goal ID	15
Goal Name	CSG_ManageOnlineOffers
Type	Composite

Continued on next page

Table 5.4 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Description	A system goal to manage online offers, the management includes removing any offer that becomes invalid, update any offer that changes in price or the quantity available to sell, and add new offers once they become available in stock.
Super Goals	GSG_ManageOnlineStore
Sub Goals	CSG_RemoveInvalidOffers, CSG_UpdateActivOffers, CSG_AddNewOffers
Fulfilled By	CBP_ManageOnlineOffers

Table 5.4: *MSMAS System Goal Descriptor: CSG_ManageOnlineOffers*

The creation of the System Goals Model, marks the end of the analysis phase. Our next step is to give an organisational structure to our system and create the system participants through the creation of Institutional Roles Models.

Institutional Roles Models

In this section we present the visual design and properties of an institutional roles model based on our case study. In this simple example we think of our system as an open society where some foreign agents that are not company-recognised and not developed by the company can join and operate, So a restriction on them should be in place to stop them playing the accessing the central stock data. In this institution, we have defined three invitational roles SP_SupplierAgent, SP_CentralStockManager, and SP_TranslationService. We also define the relationship between the supplier agent and other system agents, where the system participant that plays the supplier agent role, cannot play at any time the SP_CentralStockManager or the SP_TranslationService roles, these relations are put to address the restriction requirement. This restriction is applied by using the Disjoint Roles system norm.

Table 5.5 shows the full description of properties IM_VirtualStockInstitution institution model, while Figure 5-11 shows the graphical model with its three roles and two role/role relations.

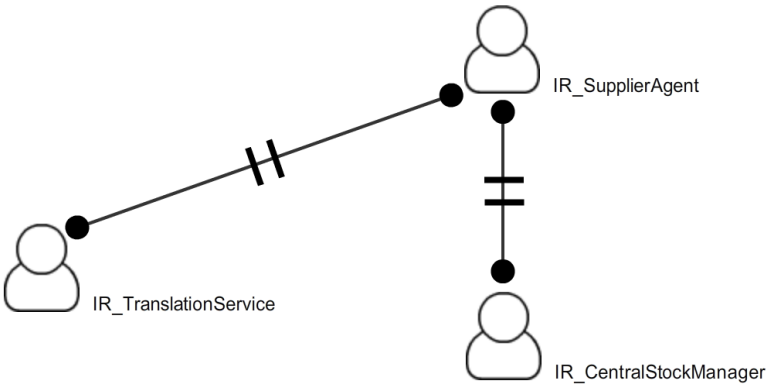


Figure 5-3: *Institutional Roles Model I: Availability Institution*

Case Study	**Virtual Stock Control and Offers Management System**
Model Name	**Institution Model I: Availability Institution (IM_AvailabilityInstitution)**
Description	Studying the system requirements reveals that there will be one or more agent per supplier, such as when there is more than one account, as a means to capture the different rules under which each account is operated. For example, one supplier might have two supplier agents with different business logic for each – reflecting different contracts – and each with access to different (physical) warehouses, affecting product availability and geographical spread for delivery. The supplier agent might: • Communicate with other suppliers' agents and exchange information about its stock levels. • Submit stock levels updates to the central stock manager. • Use one or more of the system helpers services, such as the translation service available to supplier agents to translate their specific format to the system standard format.
List of Roles	The current institution should have the following three roles: • **IR_SupplierAgent:** A role available for the supplier agents to play, that allows for exchanging information with other supplier agents and the system helpers agents/services. • **IR_TranslationService:** One of the system helper agents that offers services to facilitate exchanging information between the supplier agents and the central stock manager. • **IR_CentralStockManager:** One of the system core agents that is responsible for keeping track of the virtual central stock.

Continued on next page

Table 5.5 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
List of Role/Role Relations	• (Disjoint Roles Relation, IR_SupplierAgent, IR_CentralStockManager) • (Disjoint Roles Relation, IR_SupplierAgent, IR_TranslationService)
Roles Count	3
Created on Date	03113013
Created on Time	141010

Table 5.5: *Institutional Roles Model I: Availability Institution*

Each Institutional role created has to be fully described. Table 5.6 is an example of an institutional role descriptor. Notice that the roles have neither been assigned to any system participant nor to any business activities yet.

Case Study	**Virtual Stock Control and Offers Management System**
Name	**Institutional Role Descriptor**
Role ID	30
Role Name	IR_SupplierAgent
Description	A role available for the supplier agents to play, that allows for exchanging information with other supplier agents and the system helpers agents/services.
Member of Institution	IM_AvailabilityInstitution
Played By	Not yet assigned.
Role/Role Relations	• (Disjoint Roles Relation, IR_SupplierAgent, IR_CentralStockManager) • (Disjoint Roles Relation, IR_SupplierAgent, IR_TranslationService)
Responsible for Activity	Not yet assigned.

Table 5.6: *MSMAS Institutional Role Descriptor: IR_SupplierAgent*

Once we have created all required institutions, we can move on to the next step the refining of the system goals through adding goal/goal norms to the Composite Business process models.

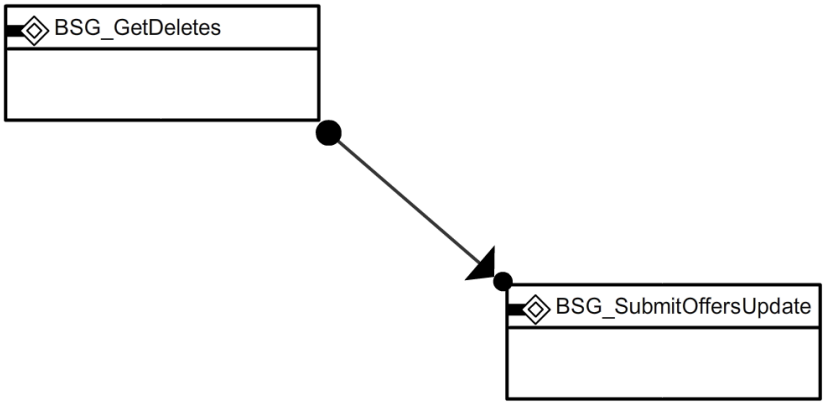


Figure 5-4: *Composite Business Process Model I (CBC_RemoveInvalidOffers) with goal/goal System Norm*

Composite Business Processes Models

In MSMAS, each composite system goal should be presented by a composite business process model, where the each subgoal of that composite system goal is presented by a sub business process within this composite business process model. To refine the relationships between these subgoals we add system norms of the type goal/goal relations. In this section we present an example based on our case study where there is a need to enforce an order of execution. Figure 5-4 shows the CBP_RemoveInvalidOffers composite business process model where we used the joint goal/goal relation between BSG_GetDeletes and BSG_SubmitOffersUpdate; this means every time the system tries to achieve the goal BSG_SubmitOffersUpdate, it is expected that it has achieved the goal BSG_GetDeletes.

Tables 5.7 shows details and properties of CBC_RemoveInvalidOffers composite business process model.

Case Study	**Virtual Stock Control and Offers Management System**
Model Name	**Composite Business Process Model: CBC_RemoveInvalidOffers**
Description	This business process presents the composite system goal CSG_RemoveInvalidOffers. It contains two subgoals: one for the supplier agent to get the list of out of stock items (deletes) and the other to submit this lit to the appropriate selling channels to be removed. To specify that each supplier agent has to check for deletes before submitting the updates we set a system norm between the two subgoals of the type joint goals, which require that BSG_SubmitOffersUpdate goal must be achieved after every occasion that BSG_GetDeletes goal is achieved.
	Continued on next page

Table 5.7 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
List Sub Business Processes	The current composite business process contains two sub processes: • **BBP_GetDeletes:** A basic business process presenting the basic system goal BSG_GetDeletes. • **BBP_SubmitOffersUpdate:** A basic business process presenting the basic system goal BSG_SubmitOffersUpdate.
List of Goal/Goal Relations	• (Joint Goals Relation, BSG_GetDeletes, BSG_SubmitOffersUpdate)
Note	The Joint goal/goal relation between BSG_GetDeletes, BSG_SubmitOffersUpdate could be done at lower level e.g. at the activity level, it is up to the system designer to choose the appropriate place for the system norms.
BPs Count	2
Created on Date	03113013
Created on Time	143550

Table 5.7: *Composite Business Process Model: CBP_RemoveInvalidOffers*

Once we have refined all goal/goal relations, we can to the detailed design of our business processes. The following section present an example of the basic business process model.

Basic Business Process Models

Let us now consider this scenario based on our case study, where the supplier agent needs to publish its stock updates by sending a file in the standard SSLF format to the central stock manager agent. Supplier agents that get their update file in their own custom format may use the translation service to translate their stock update file into SSLF format. Supplier agents can download the stock file from an FTP server or from a mail server.

Figure 5-5 shows the basic business process model of BBP_PublishSupplierStock basic business process. In this model we have created five business activities and one activity/activity relation. The *entry* business activity is BA_GetConnection, which has no precondition set, and the *exit* activity is BA_PublishSupplierStock. Details and properties of this model is shown in Table 5.8 while the descriptor of BA_GetConnection and BA_TranslateFile activities are shown in Table 5.9 and Table 5.10.

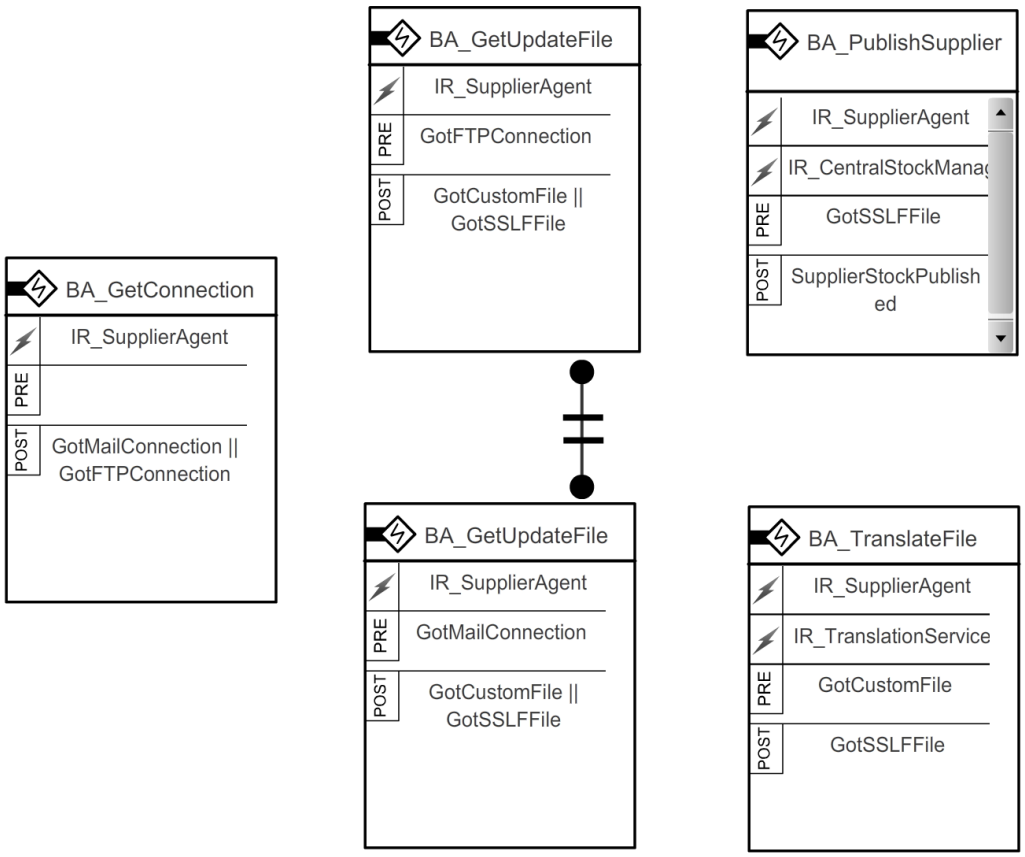


Figure 5-5: *Basic Business Process Model I (BBP_PublishSupplierStock) with System Norms*

Case Study	**Virtual Stock Control and Offers Management System**
Model Name	**Basic Business Process Model: BBP_PublishSupplierStock**
Description	This business process presents the basic system goal BSG_PublishSupplierStock. It contains five activities and one activity/activity relations. The purpose of this business process is to allow the supplier agent to publish its stock updates through sending a file in the standard SSLF format to the central stock manager agent. Supplier agents that get their update file in their own custom format may use the translation service to translate their stock update file into SSLF format. Supplier agents can download the stock file from an FTP server or from a mail server. The entry activity of this business process is BA_GetConnection and the exit activity is BA_PublishSupplierStock.
	Continued on next page

Table 5.8 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Business Activities	The current basic business process contains five activities: • **BA_GetConnection:** Entry business activity, no pre-conditions. • **BA_GetUpdateFile:** requires GotFTPConnection belief. • **BA_GetUpdateFile:** requires GotMailConnection belief. • **BA_TranslateFile:** requires GotCustomFile belief. • **BA_PublishSupplierStock:** Exit business activity, requires GotSSLF belief.
List of Activity/Activity Relations	• (None Co-existance Relation, BA_GetUpdateFile(212[1]), BA_GetUpdateFile(211))
Note	In the formal presentation we use both the activity name as well as its id to distinguish between activities that have the same name but different pre-conditions.
BA Count	5
Verified	False
Created on Date	03113013
Created on Time	145510

Table 5.8: *Basic Business Process Model: BBP_PublishSupplierStock*

Case Study	**Virtual Stock Control and Offers Management System**
Business Activity Name	**Basic Activity Process: BA_GetConnection**
Business Activity ID **Description**	77 This business activity is used to allow the supplier agent to obtain network connection to either a FTP server or email server depending on its capability and configurations.
Responsibility of	IR_SupplierAgent
List of Activity/Activity Relations	None
Required Communication protocols	None
Partially Fulfills Goal	BSG_PublishSupplierStock

Continued on next page

[1] An internal id set by the designer tool to easily identify business activities with the same name

Table 5.9 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Pre-Conditions	None
Post-Conditions	• GotMailConnection OR • GotFTPConnection
Parent Business Process	BBP_PublishSupplierStock

Table 5.9: *Basic Activity Descriptor: BA_GetConnection*

Case Study	**Virtual Stock Control and Offers Management System**
Business Activity Name	**Basic Activity Process: BA_TranslateFile**
Business Activity ID	80
Description	This business activity is used to allow the supplier agent to translate its custom stock file into SSLF by utilising the system translation service.
Responsibility of	• IR_SupplierAgent • IR_TranslationService
List of Activity/Activity Relations	• (Response Relation, BA_TranslateFile, BA_ReportEvent)
Required Communication protocols	Not Yet Assigned
Partially Fulfills Goal	BSG_PublishSupplierStock
Pre-Conditions	GotCustomFile
Post-Conditions	GotSSLF
Parent Business Process	BBP_PublishSupplierStock

Table 5.10: *Basic Activity Descriptor: BA_GetConnection*

The designing of the basic business process models highlights the needed communication protocols, where a communication act is required wherever there is more than one institutional role responsible for one business activity. In the following section we present one communication protocol as an example from our case study.

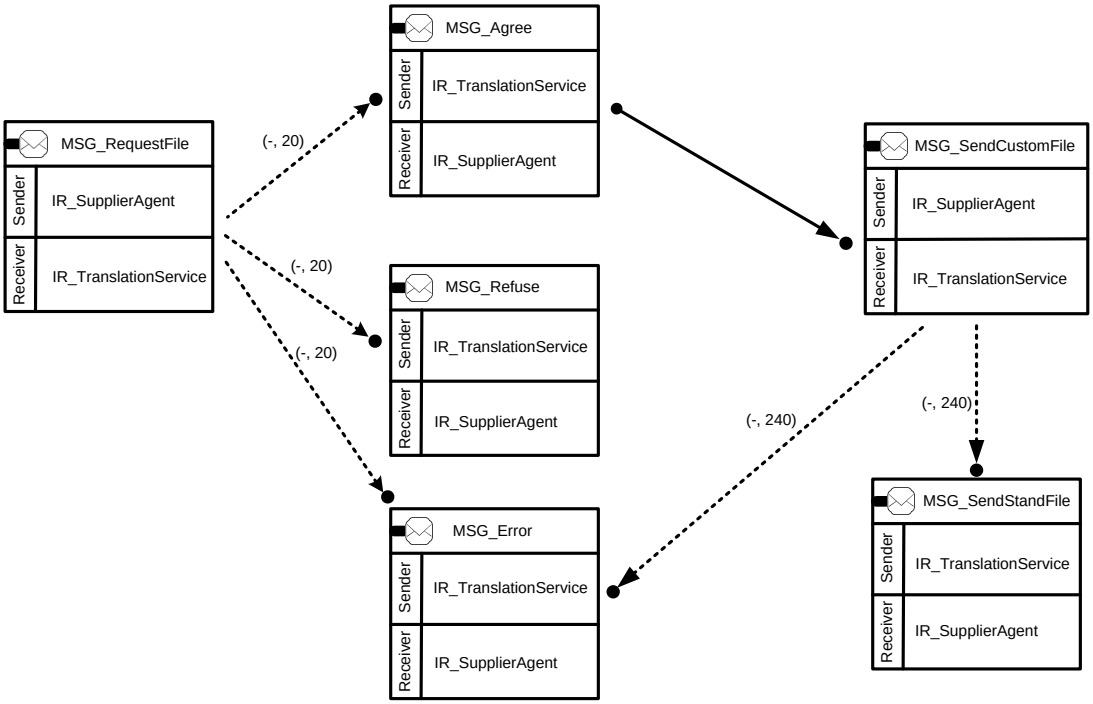


Figure 5-6: *Custom Communication Protocol Model (CCP_TranslateFile)*

Communication Protocols

Noticing the business activity BA_TranslateFile 5.10 in our BBP_PublishSupplierStock basic business process model example 5-5 shows that there are two Institutional Roles responsible for this activity IR_SupplierAgent and IR_TranslationService. Hence we need a communication protocol to facilitate their interaction. We have designed a custom communication protocol CCP_TranslateFile as shown in Figure H-1. The protocol has six messages and six message/message relations that define the acceptable message flow sequence. All possible execution routes of this protocol are shown in the AMUL diagram shown in Figure 5-7, while Table 5.11 shows the model descriptor.

Case Study	**Virtual Stock Control and Offers Management System**
Model Name / CP Name	**CP_TranslateFile**
CP ID	115
Description	A communication protocol used by supplier agents to communicate with the translation service to send the supplier custom file and receive the file translated into SSLF format.
Used in	BA_TranslateFile
Used by	• **Sender:** IR_SupplierAgent • **Receiver:** IR_TranslationService

Continued on next page

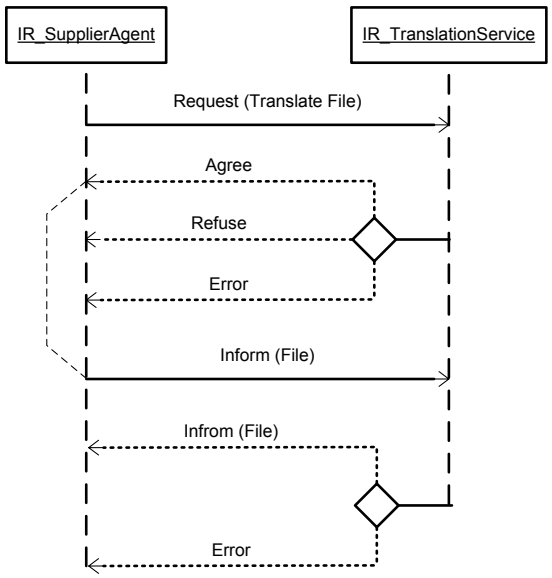


Figure 5-7: *Visual Representation of (CP_TranslateFile) Communication protocol*

Table 5.11 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Message IDs	1. (501_RequestFile) 2. (502_Agree ‖ 503_Refuse ‖ 504_Error) 3. (505_Inform) 4. (506_Inform ‖ 507_Error)
CP Type	Custom.
List of Message/Message Relations	• (Sequential Messages Relation, MSG_RequestFile, MSG_Agree) *OR* (Sequential Messages Relation, MSG_RequestFile, MSG_Refuse) *OR* (Sequential Messages Relation, MSG_RequestFile, MSG_Error) • (Joint Messages Relation, IR_Agree, IR_SendCustomFile) • (Sequential Messages Relation, MSG_SendCustomFile, MSG_Error) OR (Sequential Messages Relation, MSG_SendCustomFile, MSG_SendStandFile)
Message Count	6
Created on Date	03113013
Created on Time	150510

Table 5.11: *MSMAS Communication Protocol Descriptor: CP_TranslateFile*

This protocol is to be used by the supplier agents that need to translate their custom update file into the SSLF standard format. In our case study, the SSLF is the only format recognised and accepted by the central stock manager and supplier agents have to use it to be able to submit their updates. In this protocol, *CP_TranslateFile*, there are a maximum of four steps and a minimum of two. As shown in Figure 5-7, the communicative act starts by the supplier agent sending a *Request* message. This message could be designed to allow the supplier agent to specify the file format, the size, and number of records. The use of a *Request* message requires the translation service to respond, it has one of three options. The first is to *Agree* to the translation request, in this case the supplier agent after receiving the *Agree* message reply back with an *Inform* message that contain its file, then the translation service replies either with the file translated attached to an *Inform* message or with an *Error* message that contains a failure code indicative of the type of error it has encountered. The second option in step two can be to *Refuse* the translation request. This could be due to any reason such as if the service is too busy or it is not qualified to translate this particular custom file format. The third option in step two is to send an Error message, that normally indicates a failure and it contains a failure code.

Table 5.12 and Table 5.13 show example message descriptors from CP_TranslateFile custom communication protocol.

Case Study	**Virtual Stock Control and Offers Management System**
Communication Message Name	**CM_TransFileReq**
Message ID	501_RequestFile
Description	This message is used to initiate the translation request protocol, to be sent to one of the available translation service providers.
Used in activity	BA_TranslateFile
Used by	IR_SupplierAgent
Message/Message Relations	• (Sequential Messages Relation, MSG_RequestFile, MSG_Agree, [-;20]) *OR* (Sequential Messages Relation, MSG_RequestFile, MSG_Refuse, [-;20]) *OR* (Sequential Messages Relation, MSG_RequestFile, MSG_Error, [-;20])
Sender	IR_SupplierAgent
Receiver	IR_TranslationService
Reply to	NA
Content	FILEDESC
Language	CUSTOM
Protocol	CP_TranslateFile
Conversation-id	TBD
Reply-with	TBD

Continued on next page

Table 5.12 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
In-reply-to	NA
Reply-by	NA

Table 5.12: *MSMAS Communication Message Descriptor: CM_TransFileReq*

Case Study	**Virtual Stock Control and Offers Management System**
Communication Message Name	**CM_AcceptTransFileReq**
Message ID	502_Agree
Description	This message to respond positively to a translation request protocol, to be sent to supplier agent that requested its file to be translated.
Used in activity	BA_TranslateFile
Used by	IR_TranslationService
Message/Message Relations	• (Joint Messages Relation, MSG_AcceptTransFileReq, MSG_SendCustomFile)
Sender	IR_TranslationService
Receiver	IR_SupplierAgent
Reply to	NA
Content	(Response=True)
Language	CUSTOM
Protocol	CP_TranslateFile
Conversation-id	TBD
Reply-with	TBD
In-reply-to	NA
Reply-by	NA

Table 5.13: *MSMAS Communication Message Descriptor: CM_AcceptTransFileReq*

Once we have created all required communication protocols, can finalise our system design through the creation of Basic Roles/Activities Models.

Basic Roles/Activities Models

Basic Roles/Activities Model shows (BRAM) the same context of basic business process model from the institutional role perspective, we present in this subsection BBP_PublishSupplierStock Basic Roles/Activities Model as shown in Figure 5-15.

In this model, the entry business activity is BA_GetConnection; which has no preconditions, the exit activity is BA_PublishSupplierStock which has the postcondition that matches the system goal associated with this business process. In this model there are four institutional roles (IR_SupplierAgent, IR_TranslationService, and IR_CentralStockManager) executing four

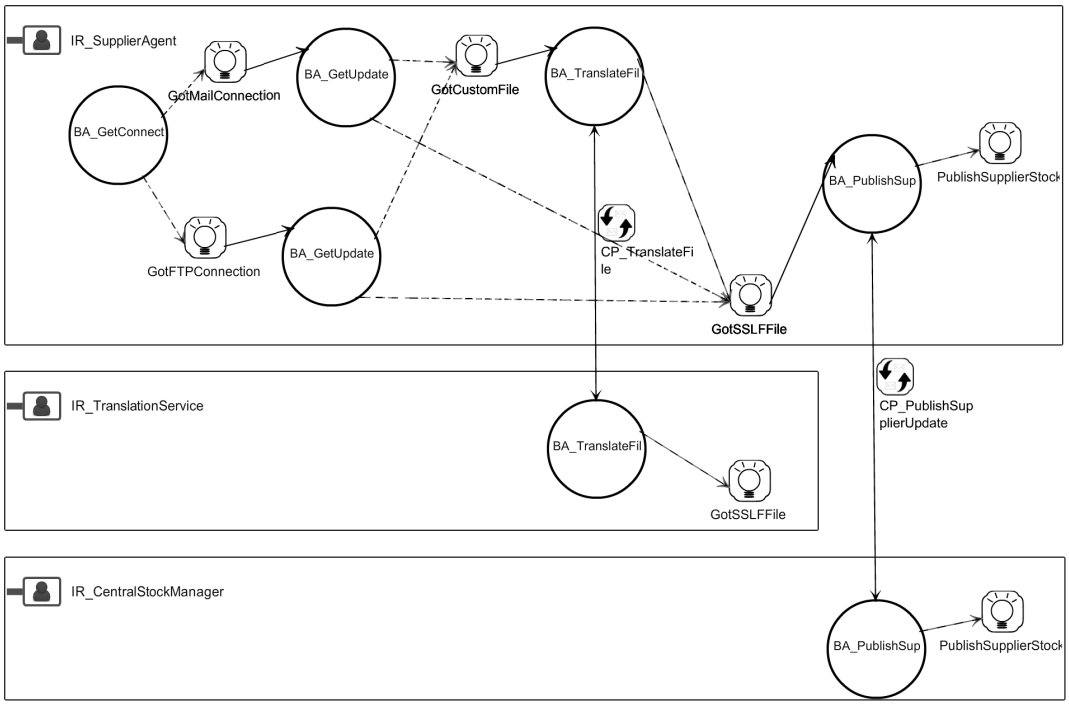


Figure 5-8: *Basic Role/Activity Model (BBP_PublishSupplierStock)*

possible business activities (BA_GetConnection, BA_GetUpdateFile, BA_GetUpdateFile[2], BA_ReportEvent, BA_TranslateFile, BA_PublishSupplierUpdate). Two of these activities (BA_TranslateFile, BA_PublishSupplierUpdate) require the system participants that play these institutional roles to interact using custom communication protocols (CCP_TranslateFile, CCP_PublishSupplierUpdate). The arrows connecting each activity with beliefs show the precondition (the arrow head points to the activity) and the postconditions (the arrow head points to the belief). Dotted arrows indicate optional conditions. For example the postcondition of BA_GetConnection is either *GotMailConnection* or *GotFTPConnection* belief, similarly the postcondition of BA_GetUpdate is either *GotSSLF* or *GotCustomFile*.

Table 5.14 shows the descriptor of BRAM_PublishSupplierStock Basic Roles/Activities Model:

[2] This activity is an example of using overloading feature, where two activities have same name but each requires different inputs, the preconditions of this activity is GotMailConnection while the other activity's precondition is GotFTPConnection

Case Study	**Virtual Stock Control and Offers Management System**
Model Name	**Basic Roles/Activities Model: BRAM_PublishSupplierStock**
Description	This business process presents the basic system goal BSG_PublishSupplierStock. The supplier agent can download the stock file from an FTP server or from a mail server, depending on the file type the supplier agent either translate it to the SSLF format or submit it directly to the central stock manager. The entry activity of this business process is BA_GetConnection and the exit activity is BA_PublishSupplierStock.
Business Activities	The current basic business process contains five activities: • **BA_GetConnection:** Entry business activity, no pre-conditions. • **BA_GetUpdateFile:** requires GotFTPConnection. • **BA_GetUpdateFile:** requires GotMailConnection. • **BA_GetTranslateFile:** requires GotCustomFile. • **BA_PublishSupplierStock:** Exit business activity, requires GotSSLF.
List of Activity/Activity Relations	• (None Co-existance Relation, BA_GetUpdateFile, BA_GetUpdateFile)
Institutional Roles	• **IR_SupplierAgent** • **IR_TranslationService** • **IR_CentralStockManager**
Communication Protocols and Messages	• **CP_TranslateFile** • **CP_PublishSupplierUpdate**
Verified	False
Created on Date	03113013
Created on Time	145510

Table 5.14: *Basic Business Process Model: BRAM_PublishSupplierStock*

In the last subsections, we have presented example models of a system based on real world application scenario. These models do not present the complete set of designs to model this system, however they give a clear view of how to conduct each step of MSMAS methodology and what are the expected artifacts following each step. We have kept intentionally these examples simple to assist the reader in understanding and focusing on the nature of each step and the use of visual notations and descriptors. In the following sections we will show how some of these models can be advanced through the inclusion Self-Managing features in our system.

5.2 Example Models for Self-management

Let us now illustrate how can we model and introduce self-managing mechanism to the models of our case study system. Consider this set of new requirements:

1. The system should manage its components and new system participants have to register with an admin entity before being able to actively participate in any activity.
2. All system participants has to report about their interactions with other system participants including the system helpers.
3. An admin entity has to check for violation and to suspend any violating system participant.
4. An admin entity has to replace any faulty system helper with a instance.
5. Monitoring and admin entities should be company owned components.
6. A monitoring entity can not become an admin before 100 time units.

To introduce these new requirements to our system model, let us first create a new use case to highlight the needed set of institutional roles and the new set of needed system goals.

Use Case Model

In the "Suspend Non-compliant Supplier Agent" we show that every time the supplier agent interacts with the central stock manager, it has to submit a report about this interaction to a new system participant a monitoring agent. The monitoring agent then checks these event reports against the system norms model. If there is a match, then the monitoring agent just create a log entry, but if the event matches one of the unexpected events, the monitoring agent interprets this as a violation of system norms and notifies an admin agent, which in turn applies the chosen corrective action by suspending the violating supplier agent, so it can no longer access or interact with any other agent. This implementation is inspired by the work of Roy [2007b] on complex feedback loops as a method to monitor any system. In MSMAS we can generate a formal model for the system specifications in terms of desired and undesired events. We can use this model as a reference of predicted system states.

Figure 5-9 shows the visual model of this use case while Table 5.15 shows the details and properties of this use case.

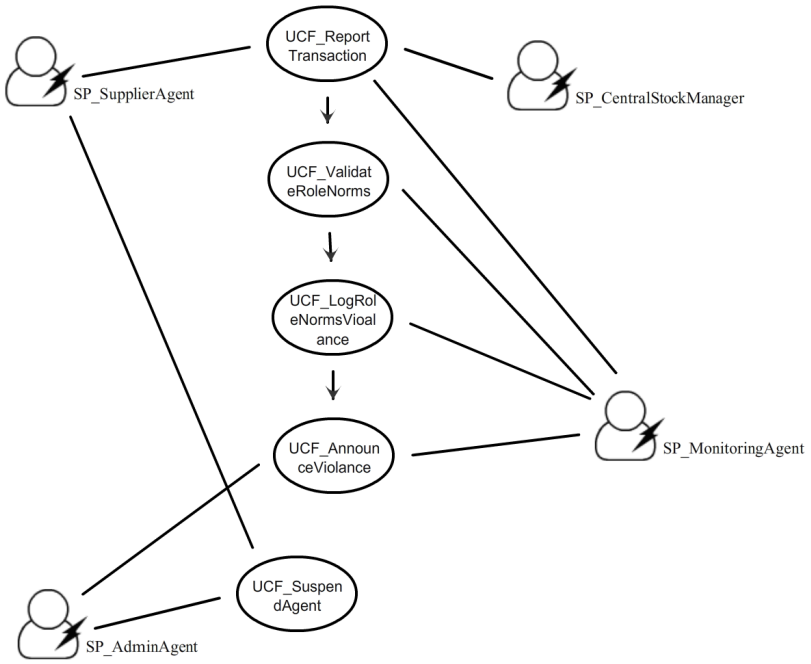


Figure 5-9: *Use Case Model II: Suspend Non-compliant Supplier Agent*

Case Study	**Virtual Stock Control and Offers Management System**
Use Case ID	03
Use Case Name	**Use Case II: Suspend Non-compliant Supplier Agent**
Description	This use case shows how a monitoring agent could be used to detect a violation of system norms and how the admin agent can apply corrective action by suspending the violating supplier agent. In this case study, we require each system participant to report about each interaction with other system participants, so any interaction is seen as a sub-system and each participating agent is part of the first component (monitoring) that reports about the state of the other agent. The monitoring agent in this use case is the second component, the one that receives all reports and calculates the actions to check for any violation. The Admin agent is the third component that applies the corrective action once a violation is detected.
Goals	1. Monitor supplier agent actions (a) Validate the action against the system norms. (b) Log any detected violation. (c) Notify the admin agent of each case of violation. 2. Manage Supplier Agent membership (a) Register new supplier agents (b) Suspend supplier agent
Preconditions	Transaction Report Received

Continued on next page

Table 5.15 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
System Participants	There are four different system participants in this use case: 1. **SP_MonitoringAgent:** a system participant that is responsible for receiving reports from all other system participants about their interactions with one another. 2. **SP_SupplierAgent:** an agent that presents one actual supplier in the current system and is responsible for providing accurate information about the supplier physical stock levels and cost. 3. **SP_AdminAgent:** is the governor of the system, the agent with the ultimate power that enforces the system norms upon all participating agents. 4. **SP_CentralStockManager:** system participant that is responsible for the managing of the central stock where the consolidated quantities per each item available from all suppliers that can supply these items are stored. It is responsible for keeping this virtual stock as accurate as possible and close to real world physical stock, to avoid overselling some items and loss of synchronisation.

Continued on next page

Table 5.15 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
System Functions	As shown in Figure 5-9 we have identified five functions: 1. **UCF_ReportTransaction:** a function to be used by all system participants when they interact with other system participant, to report about that interaction. [3] When two system participants interact, their interaction is seen as a sub-system and each one of them will report about the state of the other system participant to the monitoring agent using this function. 2. **UCF_ValidateAgainstSystemNorms:** a function that each monitoring agent uses to check if the stream of received actions matches the expected course of actions for each system participant playing a particular role. The system norms are presented as a formal model in terms of desired and undesired events. A violation is detected if the monitoring agent receives a report of one or more of the undesired actions. 3. **UCF_LogRoleNormsViolence:** Logging function that allows the monitoring agents to log each violation they detect. We have chosen to keep track of each violation, then we can implement various responses based on on the number of violations per agent and their frequency. 4. **UCF_AnnounceViolence:** A function used by the monitoring agent to notify the admin agent of any violation it has detected. In this implementation, the monitoring agent is the second component of the feedback loop, the component that calculates and detect the violation if any, and the admin agent is the third component that applies the corrective action. 5. **UCF_SuspendAgent:** A function that allows the admin agent to isolate any non-compliant agent.
Normal Flow	• Each agent reports about every transaction with another agent to the monitoring agent. • The monitoring agent validates the action of each agent against the system norms. • The monitoring agent logs any detected violation of system norms. • The monitoring agent notifies the admin agent about each violation. • The admin agent applies the appropriate corrective action. In this use case it suspends the supplier agent.

Continued on next page

[3] This is part of the self-management mechanism implemented as explained in [Elakehal and Padget, 2008] using a feedback loop, where we have three components: one that monitors the state of a sub-system, another that calculates the corrective action, and a third component that applies the corrective action.

Table 5.15 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Exceptions	• Transaction report received from only one system participant. • Monitoring agent does not respond. • Admin Agent does not respond.
Notes	None

Table 5.15: *Use Case II: Suspend Non-compliant Supplier Agent*

System Goals Model

Now we revisit the system goals model to include the new set of system goals to satisfy the new set of business requirements. We include the following list of goals: GSG_ManageOnlineStore

1. ...
2. ...
3. CSG_ManageSystemComponents
   (a) CSG_ManageSuppliersMembership
      i. BSG_RegisterSupplierAgent
      ii. BSG_SuspendSupplierAgent
      iii. BSG_SubmitEventsReport
   (b) CSG_ManageSystemHelpers
      i. BSG_AddSystemHelper
      ii. BSG_RemoveSystemHelper
   (c) CSG_MonitorSystemComponents
      i. BSG_ReceiveEventsReport

Figure 5-10 shows the modified version of the system goals system model after the inclusion of these set of goals.

Institutional Roles Models

With the introduction of the new system roles and system goals in the use case and the system goals model, we need to revisit the "Availability Institution" to modify it to respond to this open world structure. We add two new intuitional roles IR_AdminAgent and IR_MonitoringAgent. In this modified model, we also added new system norms to restrict the ability of a system participant playing some roles at the same time or playing a specific role after it has played another role. Based on the system requirements, we want to *forbid* any foreign agent (an agent developed by a third party) joining the system such as a IR_SupplierAgent to play the role of IR_AdminAgent or IR_CentralStockManager. We set a disjoint role/role relation between IR_SupplierAgent and IR_AdminAgent and another disjoint role/role relation between

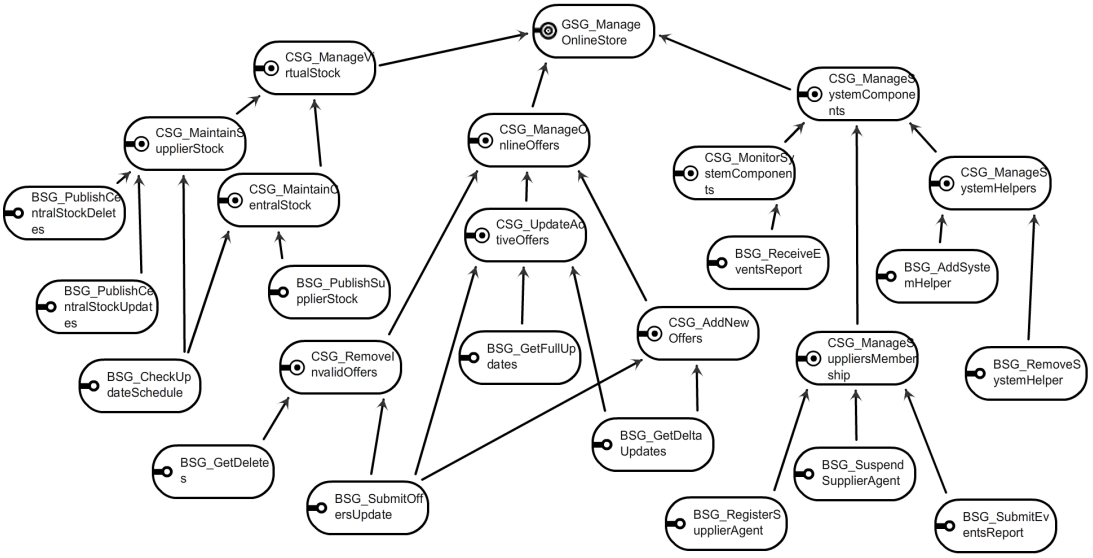


Figure 5-10: *Modified Version of System Goals Model: Virtual Stock Control and Offers Management System*

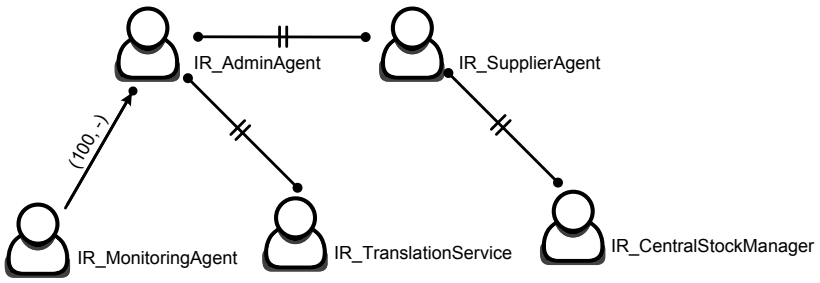


Figure 5-11: *Modified Version of Institutional Roles Model I: Availability Institution*

IR_SupplierAgent and IR_CentralStockManager. Similarly to respond to the system requirements to forbid and agent playing IR_TranslationService role from playing the IR_AdminAgent role we set a disjoint role/role relation between IR_AdminAgent and IR_TranslationService. The last requirement of the institution is to require that any agent wants to play the IR_AdminAgent has to play IR_MonitoringAgent role beforehand for at least 100 time unites. We set a sequential role/role relation with a quantitative time constraint to specify this.

We also create a second institution IM_VirtualStockInstitution, in this institution we relax the restrictions because no foreign agents are expected to join this system segment. Figure 5-12 shows the graphical model of this institution and Table 5.16 shows the model properties.

Case Study	**Virtual Stock Control and Offers Management System**
Model Name	**Institution Model II: Virtual Stock Institution (IM_VirtualStockInstitution)**
Description	The function of this institution and the system participants who play its roles is to manage the online offers in different marketplaces. There is no foreign agent expected to join this system segment, instead all agents can play any of the roles in this institution and are to be designed and developed by the system owner. For this reason is less concern about the type of data exchanged and the security of communications between these agents. The main core roles are the central stock manager and the listings manager, where the former provides an accurate stock count to the latter to update the online offer accordingly. Listing updates and other stock check functions are normally scheduled tasks, so there is a need for an agent to manage the timing of the various tasks. The system still needs to manage its agents to ensure any failure is detected and any faulty agent to be replaced. For this reason the admin agent and monitoring agent roles are required in this institution.
List of Role/Role Relations	The current institution should have the following five roles: • **IR_SchedulerAgent:** A role available for one or a group of the system agents to mange the timing of various update tasks. This role type is an amicable role. • **IR_AdminAgent:** A role for one or more system agents to play to control the membership of supplier agents and manage the helper agents as well as enforce the system norms on all members agents. • **IR_MonitoringAgent:** A role available for one or more of the system agents that allows them to receive event reports, to check these events against the system norms to detect any violation and to notify the admin agents of any violation incident. A violation in this institution might be that one of the system agents is not responsive any more, and the corrective action the admin agent could take then is to replace the faulty agent with another system agent. • **IR_ListingsManager:** A role that is available for one of more of the system agents that is responsible for managing the online offers in one or more marketplaces. It is responsible upload files to the marketplace with new offers, updates of current offer, or deletes of one or more of the active offers. • **IR_CentralStockManager:** One of the system core agents that is responsible for keeping track of the virtual central stock.

Continued on next page

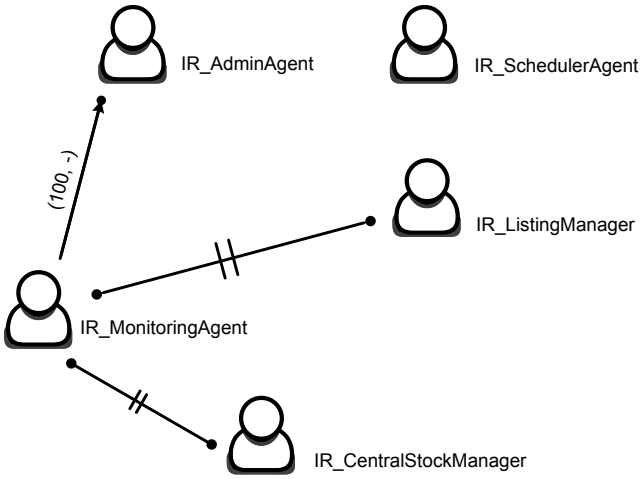


Figure 5-12: *Institutional Roles Model II: Virtual Stock Institution*

Table 5.16 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
List of Role/Role Relations	• (Disjoint Roles Relation, IR_ListingsManager, IR_MonitoringAgent) • (Disjoint Roles Relation, IR_CentralStockManager, IR_MonitoringAgent) • (Sequential Roles Relation, IR_AdminAgent, IR_MonitoringAgent, [100; -])
Note	The Disjoint role/role relations between IR_CentralStockManager and IR_MonitoringAgent and IR_ListingsManager and IR_MonitoringAgent, and the sequential relation between IR_MonitoringAgent and IR_AdminAgent implicitly guarantee that neither and agent playing IR_ListingsManager role nor any agent playing IR_CentralStockManager can play the IR_AdminAgent role.
Roles Count	5
Created on Date	03113013
Created on Time	143010

Table 5.16: *Institutional Roles Model II: Virtual Stock Institution*

Composite Business Process Models

We present now one composite business process model that shows how we can set a goal/-goal norm to enforce the requirement of submitting a report by the supplier agent after every time it sends its update file to the central stock manager. Figure 6-6 shows how we can associate a system norm with the system goals using joint goal/goal relation. This relation means

that (PublishSupplierStock) must be executed before (SubmitEventsReport) and every time the agents achieve the goal (PublishSupplierStock) it must achieve the goal (SubmitEventsReport). Table 5.17 shows the model descriptor of CBC_MaintainCentralStock composite business process model while Table 5.17 shows the description of BBP_SubmitEventsReport basic business process. Notice that once a relationship is established between two business processes within a composite business process model, a system norm is created from the type goal/goal relation and both the business process descriptor and the goal descriptor are updated accordingly.

Case Study	**Virtual Stock Control and Offers Management System**
Model Name	**Composite Business Process Model: CBC_MaintainCentralStock**
Description	This business process presents the composite system goal CSG_MaintainCentralStock. It contains two subgoals: one for the supplier agent to publish its stock updates and the other to submit event reports to help the monitoring mechanism to function. To specify that each supplier agent has to submit an event report we can set a system norm between they two subgoals of the type joint goals, which require that BSG_SubmitEventsReport goal must be achieved after every occasion that BSG_PublishSupplierStock goal is achieved.
List Sub Business Processes	The current composite business process contains two sub processes: • **BBP_PublishSupplierStock:** A basic business process presenting the basic system goal BSG_PublishSupplierStock. • **BBP_SubmitEventsReport:** A basic business process presenting the basic system goal BSG_SubmitEventsReport.
List of Goal/Goal Relations	• (Joint Goals Relation, BSG_PublishSupplierStock, BSG_SubmitEventsReport)
Note	The Joint goal/goal relation between BSG_PublishSupplierStock, BSG_SubmitEventsReport could be done at lower level e.g. at the activity level, it is up to the system designer to choose the appropriate place for the system norms.
BPs Count	2
Created on Date	03113013
Created on Time	143550

Table 5.17: *Composite Business Process Model: CBC_MaintainCentralStock*

Figure 5-13: *Composite Business Process Model I (CBC_MaintainSupplierStock) with goal/goal System Norm*

Case Study	**Virtual Stock Control and Offers Management System**
Business Process Name	**Basic Business Process: BBP_SubmitEventsReport**
Business Process ID	56
Type	Basic Business Process
Fulfills Goal	BSG_SubmitEventsReport
Description	A basic business process presents activities needed to achieve BSG_SubmitEventsReport system goal. The supplier agent reports each event needed for the monitoring process using this BP.
List Super Business Processes	The current basic business process is has two super processes: • **CBC_MaintainSupplierStock:** A Composite business process presenting the composite system goal CSG_MaintainSupplierStock. • **CBC_ManageSuppliersMembership:** A Composite business process presenting the composite system goal CSG_ManageSuppliersMembership.
List of Goal/Goal Relations	• (Joint Goals Relation, BSG_PublishSupplierStock, BSG_SubmitEventsReport)

Table 5.18: *Business Process Descriptor: BBP_SubmitEventsReport*

Basic Business Process Models

Under the new requirement we need to revisit the BBP_PublishSupplierStockBasic basic business process model, the model now has six business activities and five activity/activity relations. Figure 5-14 shows the modified visual model and Table 5.19 shows its properties.

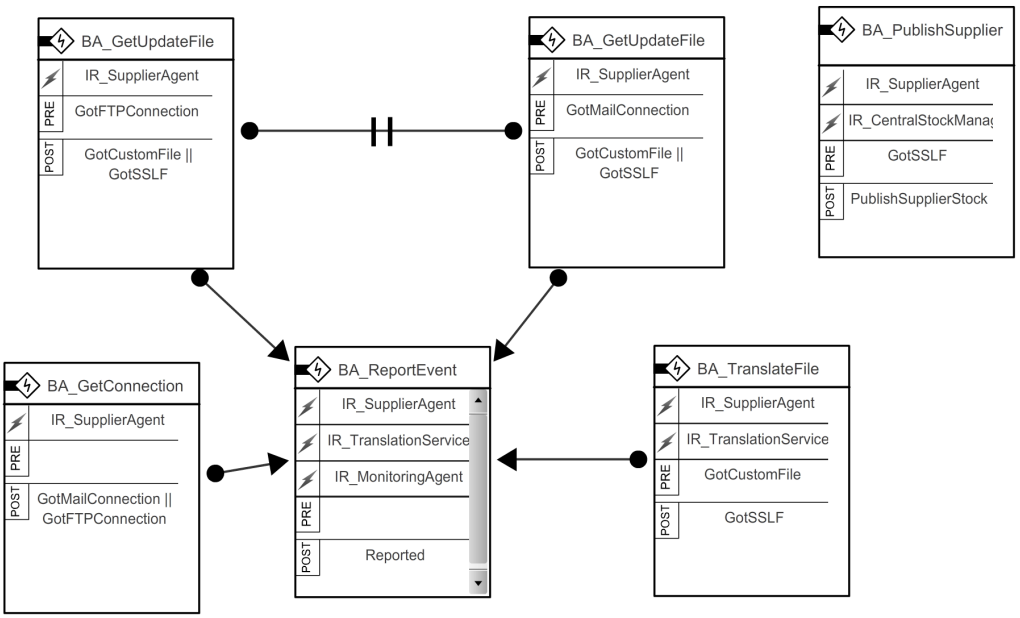


Figure 5-14: *Modified Version of Basic Business Process Model I (BBP_PublishSupplierStock) with System Norms*

Case Study	**Virtual Stock Control and Offers Management System**
Model Name	**Basic Business Process Model: BBP_PublishSupplierStock**
Description	This business process presents the basic system goal BSG_PublishSupplierStock. It contains six activities and five activity/activity relations. The purpose of this business process is to allow the supplier agent to publish its stock updates through sending a file in the standard SSLF format to the central stock manager agent. Supplier agents that get their update file in their own custom format may use the translation service to translate their stock update file into SSLF format. Supplier agents can download the stock file from an FTP server or from a mail server. Activities that are executed by more than agent are monitored by the monitoring agent. each agent has to submit an event report to the monitoring agent as soon as it starts one of these activities. To specify that each supplier agent has to submit an event report we set a system norm the activities of the type sequential activity (ConDec:Response Relation). The entry activity of this business process is BA_GetConnection and the exit activity is BA_PublishSupplierStock.
	Continued on next page

Table 5.19 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Business Activities	The current basic business process contains five activities: • **BA_GetConnection:** Entry business activity, no pre-conditions. • **BA_GetUpdateFile:** requires GotFTPConnection. • **BA_GetUpdateFile:** requires GotMailConnection. • **BA_TranslateFile:** requires GotCustomFile. • **BA_ReportEvent** • **BA_PublishSupplierStock:** Exit business activity, requires GotSSLF.
List of Activity/Activity Relations	• (Response Relation, BA_GetUpdateFile, BA_ReportEvent) • (Response Relation, BA_GetConnection, BA_ReportEvent) • (Response Relation, BA_TranslateFile, BA_ReportEvent) • (None Co-existance Relation, BA_GetUpdateFile, BA_GetUpdateFile)
Note	In the formal presentation we use both the activity name as well as its id to distinguish between activities that have the same name but different pre-conditions.
BA Count	2
Verified	False
Created on Date	03113013
Created on Time	145510

Table 5.19: *Modified version of the Basic Business Process Model: BBP_PublishSupplierStock*

Basic Business Process Models

Finally, we present a modified version of the visual model of BRAM_PublishSupplierStock Basic Business Process Model, where BA_ReportEvent business activity and IR_MonitoringAgent institutional role appear with their appropriate relations. Figure 5-15 shows the modified version of the visual model and Table 5.20 shows the model properties.

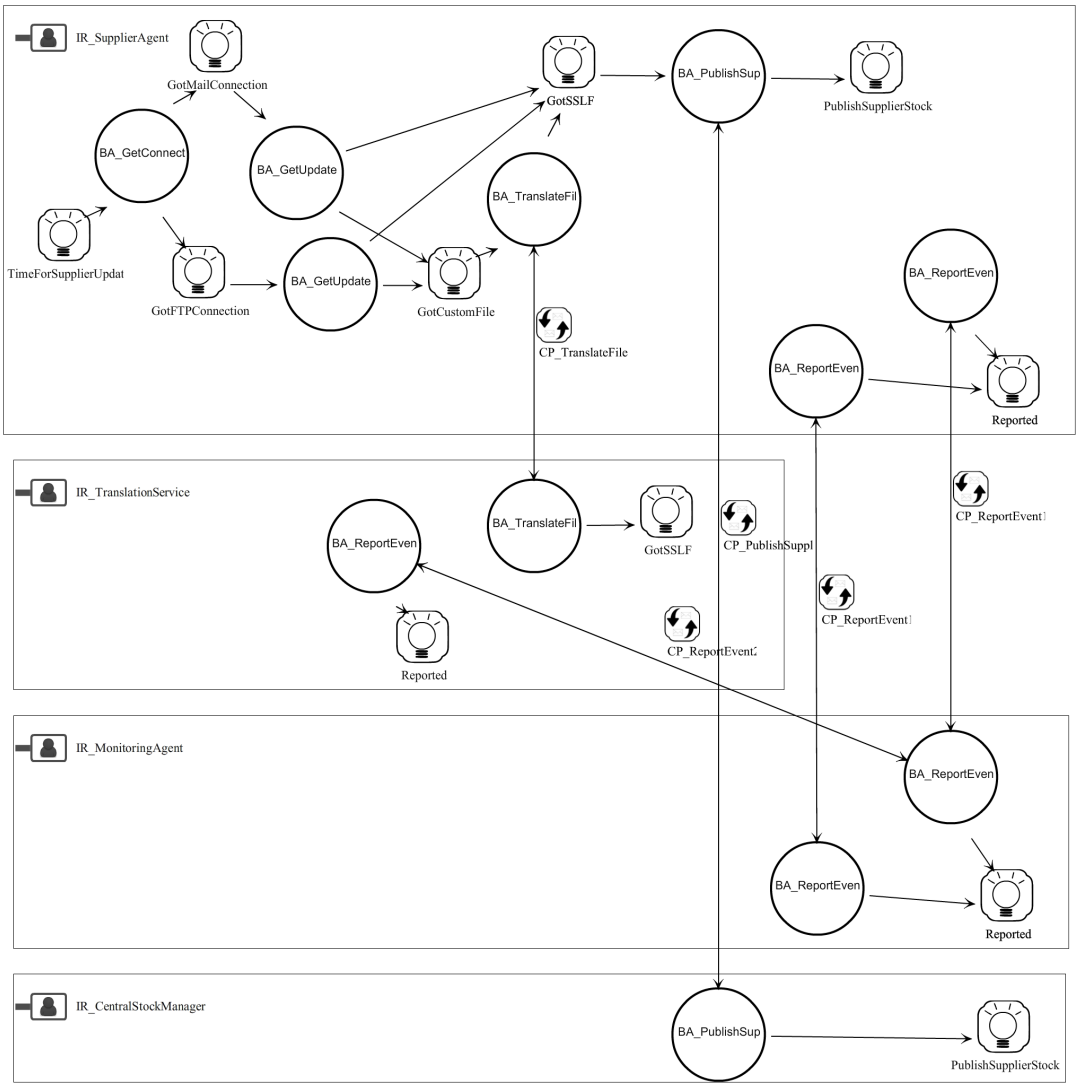


Figure 5-15: *Modified Version of BBP_PublishSupplierStock Basic Role/Activity Model*

Case Study	**Virtual Stock Control and Offers Management System**
Model Name	**Basic Roles/Activities Model: BRAM_PublishSupplierStock**
	Continued on next page

Table 5.20 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Description	This business process presents the basic system goal BSG_PublishSupplierStock. It contains six activities and five activity/activity relations. The purpose of this business process is to allow the supplier agent to publish its stock updates through sending a file in the standard SSLF format to the central stock manager agent. Supplier agents that get their update file in their own custom format may use the translation service to translate their stock update file into SSLF format. The supplier agent can download the stock file from an FTP server or from a mail server. Activities that are executed by more than agent are monitored by the monitoring agent. each agent has to submit an event report to the monitoring agent as soon as it starts one of these activities. To specify that each supplier agent has to submit an event report we set a system norm the activities of the type sequential activity (ConDec:Response Relation). The entry activity of this business process is BA_GetConnection and the exit activity is BA_PublishSupplierStock.
Business Activities	The current basic business process contains five activities: • **BA_GetConnection:** Entry business activity, no pre-conditions. • **BA_GetUpdateFile:** requires GotFTPConnection. • **BA_GetUpdateFile:** requires GotMailConnection. • **BA_GetTranslateFile:** requires GotCustomFile. • **BA_ReportEvent** • **BA_PublishSupplierStock:** Exit business activity, requires GotSSLF.
List of Activity/Activity Relations	• (Response Relation, BA_GetUpdateFile, BA_ReportEvent) • (Response Relation, BA_GetConnection, BA_ReportEvent) • (Response Relation, BA_TranslateFile, BA_ReportEvent) • (None Co-existance Relation, BA_GetUpdateFile, BA_GetUpdateFile)
Institutional Roles	• **IR_SupplierAgent** • **IR_TranslationService** • **IR_MonitoringAgent** • **IR_CentralStockManager**
Communication Protocols and Messages	• **CP_TranslateFile** • **CP_ReportEvent** • **CP_PublishSupplierUpdate**
Verified	False

Continued on next page

Table 5.20 – continued from previous page

Case Study:	**Virtual Stock Control and Offers Management System**
Created on Date	03113013
Created on Time	160510

Table 5.20: *Modified version of the Basic Business Process Model: BRAM_PublishSupplierStock*

5.2.1 Notes on the Use of Norms and Institutions

To specify the context where norms apply, norms should be *prescriptive*, so system participants have to be made aware of the rules. MSMAS norms are formalised in the way that their semantics can be shared and understood by all system participants through the sharing of their formal model. A system norm is activated when the triggering event happens, for example the restriction on playing a particular institutional role when another role is played (disjoint role/role relation) is activated for a particular system participant as soon as that system participant plays one of these roles. If the system participant plays the other role as well, it is in violation of the system norms. It is essential to make the system participants aware of their accountability if they violate the system norms. In MSMAS, we leave the handling of system norms violation to the implementation. The system designer is encouraged, however, to specify norms that are activated and trigger corrective actions that can be associated with each type of violation. We also do not specify any particular way to implement the governing authority, which is responsible for the generation, modification or abolishing system norms. Some system designers might opt for the use of a central governing body in one application, while they opt for distributed governance in another.

Designing norms should consider if they are prohibitions, obligations or permissions. The context of the violation should also determine how to handle a violation. The system designer might choose to apply a sanction for a norm violation within a particular use case and ignore the violation in another. We leave this decision to the system designer to make, depending on his/her application domain and the system requirements. The mechanism for publishing, updating and abolishing system norms is also left for the implementation, as it can be done differently depending on the implementation framework or the implementation programming language. A general rule is to let system participants be aware of the applicable norms to allow them to avoid violating the norms or at least be aware of the consequences of their violations.

An essential property of nMAS is to design followable norms, meaning that system participants can have the choice to comply with or violate the norm, and that there are no conflicting sets of norms. This particular problem of designing followable norms is addressed by checking that the designed norms do not conflict with one another [Li et al., 2012], this can be achieved through the model verification as discussed in detail in Chapter 6.

The problem of how to design a system that complies with a given set requirements is

addressed by enabling the studying of compliance at runtime through monitoring of the execution traces and checking them against the expectations as defined by the system specifications. Hence, the system requirements needs to be encoded as system norms that are articulated formally and are representing conditions under which they are triggered and applied, as defined by the first two basic features of any normative domain.

MSMAS norms can be mapped to LTL, CLIMB and Event Calculus formal presentations. The system designer can use the LTL to verify the system norms during design time, using Buchi Automata techniques or using $\mathcal{S}$CIFF proof procedure for verifying the system model. Event Calculus (EC) technique can be used to monitor the execution traces at run time as discussed in detail in Chapter 6.

MSMAS allows for specifying norms with temporal properties such as: the time when the norm comes into force (triggering event), the time when the norm can produce an effect (applicable norm), and the time when the normative effects hold (post condition) [Governatori and Rotolo, 2010]. MSMAS uses the ConDec^{++} visual notation with metric time indicators on the applicable relations. These time constrained relations are mapped to CLIMB formulae or EC theories with both qualitative and quantitative time support.

ConDec^{++} and commitments have a similar approach with regard to their underlying paradigm, as both model interactions in an open and declarative fashion, but their purpose is different. While commitments focus on the mutual obligations, characterising the (un)desired and exceptional state of affairs, while staying away from the specific event occurrences, ConDec^{++} constrains the acceptable execution traces, which makes ConDec^{++} more suitable for modelling, checking and monitoring any event-based system.

To respond to the question of how norms can be used to define the organisational design of the system participants, in MSMAS we associate the norms with institutional roles played by the system participants. These roles are members of institutions that represent a fragment of the society or a sub-organisation. Institutional roles and their relations are considered to be constitutive norms and used to grant powers to the system participants who play these institutional roles inside the organisation/institution. Playing a role is an enabler for that system participant to communicate and interact with other system participants within the organisation. Managing the assignment of system participants to institutional roles is a straightforward method to manage the organisation.

Institutions as defined by Ruiter [1997] are those "systems of regulative and constitutive rules that provide frameworks for social action within larger rule-governed settings". In MSMAS roles and communication protocols. Norms are considered constitutive norms while the norms between activities and system goals are regulative norms that define different executions paths. The definition of a shared institutional framework does also require the definition of monitoring and governing components that take care of detecting conformance or any violation and applying rewarding or corrective actions. The institution structure allows the system participants to plan the future actions that the interacting agents need, based on clear formal

expectations. This is possible in MSMAS if the system designer allows the sharing of commonly accepted sets of rules or the system norms applicable to this institution. These in effect define the obligations, prohibitions, permissions of the interacting parties. MSMAS allows for the fundamental principle of expressing institutional and other system norms using a declarative formal language because this allows the represention of these norms as data that can be queried and changed dynamically during run time if needed, instead of coding them in each software component. The possibility of adding, changing, or removing the norms that regulate the interaction both at design time or at run-time is one of the biggest advantages the MSMAS offers, where the specification of the system can be changed without the need to reprogram the interacting system participants. Add to that, the system participants can automatically reason about the consequences of their actions if the system designer embeds the rewarding actions as well as the sanction actions within the system norms. For example, if one of the violation can lead to suspension of institution membership, this could be added as a system norm requiring the system administration (governing body) to suspend the membership of the system participant if a particular violation is detected. Using an application-independent monitoring component can then enable the system to keep track of the state of norms on the basis of the happened events versus expected events.

5.3 Chapter Summary

In this chapter, we presented a a case study based on a real world application for the online selling of products. In the first part of this chapter, we presented a number of visual models and descriptors that illustrate the modeling process according to MSMAS methodology. In the second part, we focused on presenting examples that support self-management. We presented a number of new models and revisited the case study models to modify them in a way that shows how to model the system based on the discussed self-management approaches.

CHAPTER 6

MSMAS FORMAL MODELS VERIFICATION AND RUN-TIME MONITORING

6.1 Introduction

Complex system development presents a great challenge to all system developers, system modelers and system designers. Systems that have business critical functions are required to perform at a high level of accuracy and resilience. One way to build such systems is by implementing various self-management techniques. Complex systems, however, are more difficult to design and one of the biggest challenges of the design process is how to verify that the system models are a fair representation of the design goals, meaning that it will not fail to deliver on the requirements once developed and deployed.

Verifying the system model requires that model be formally described. The formal representation allows for the use of mathematical and/or logical techniques which require great efforts to develop initially but once created, can be reused cheaply for an efficient verification process. Model Verification is the process of verifying properties of a system through model checking techniques instead of testing the actual system that might be a very expensive and very time-consuming process, we can verify the system properties faster and more accurately, through automated exhaustive checking on abstracted system models.

Multiagent systems are theoretically open systems, they offer a suitable environment where the system components enjoy a high degree of flexibility and freedom in deciding on their own course of actions. In MSMAS a MAS is a system where various System Participants can interact and exchange information, human actors are one of the system participant types and take part in the system along side autonomous agents, services. The organisation structure of the system participants is an institution structure which is a specific class of multi-agent systems where agent behaviour is governed by social norms and regulations.

As mentioned earlier in Chapter 2 Section 2.6 Page 43, in open agent societies, agents

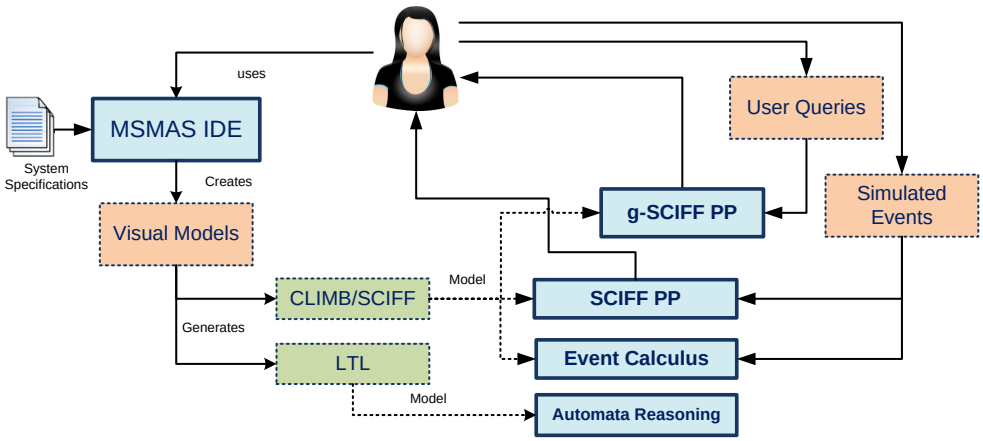


Figure 6-1: *MSMAS Model Verification Process at Design Time*

cannot be trusted to behave all the time in compliance with the main system goals. The use of the norms to model the expected behaviourial patterns then becomes essential for these normative systems. By building logic-based agents not only can we verify their models, but also, we can understand wether the execution trace matches the requirements or violates them. The system requirements are normally captured according to the formal model where they define what the agent can do or not. Restricting the system components' behaviours can be powerful, but if not done with correct measures it can turn out to be disastrous, thus model verification has great importance and becomes an integral part of the full life cycle of building MASs.

Verifying that a MAS system actually behaves as it is supposed to, can be done through testing, or a formal proof of correctness, safety and liveness. Formal verification includes theorem proving and model checking methods. In this chapter we cover the formal models of MSMAS discussed under two main sections in the context of Static Verification of these models, that is applicable during the design time (a-priori) and in the context of Monitoring and Run-time Verification (runtime).

In the following subsections we address the question of why should we use a formal model and which formalism is more suitable for representing MSMAS norms and models. In Section 6.2, we summarise the CLIMB framework, then in section 6.3 we present our mapping of one of MSMAS norm groups into CLIMB. The full mapping of the remaining norm groups can be found in Appendix H. In Section 6.5, we summarise the Event Calculus Framework, then in section 6.6 we set out our representation of one of MSMAS norms as EC theories. More example representation of the remaining norm groups can be found in Appendix I. Finally, Section 6.4 covers the static verification method with an example model, and Section 6.7 addresses the validation of a deployed system using monitoring and runtime verification with an example.

6.1.1 Why the use of formal models?

Creating flexible open system raises the problem of interoperability between autonomous components and it leads to the following two crucial characteristics [Fornara et al., 2013b]:

- No assumptions can be made about either the internal structure or the real intentions and abilities of the interacting parties especially if they are going to satisfy the rules, the norms, and the interaction protocols.
- Interacting agents need to plan their future communicative and non communicative actions: this is normally based on expectations of other agents future actions. One way to achieve this is through the belief that all other agents can reach the same conclusions from the available information. Therefore a common semantics for the meaning of the exchanged messages and system norms is needed to be in place and to be shared.

Fuchs and Robertson [1996] state that mapping an application-specific specification language to logic allows application developers not only to formulate specifications in familiar terms, but also to support and conduct formal validation and verification. Furthermore, expressing system norms formally enables the system participants to be aware of what is expected from them, and enables the inclusion of reasoning and learning within the system processes, based on checking system participants actions against the set of possible allowed actions, according to the formal model.

6.1.2 Which Logical Formalism?

Each logical framework offers a range of features that might differ from the ones offered by other frameworks. Our choice of a declarative language such as ConDec^{++} means there are a number of possibilities for providing suitable underlying semantics in terms of logic-based languages.

The first option is mapping ConDec models to LTL (on finite traces), which is a form of modal logic that uses temporal operators besides the classical logical operators. Models that are expressed in LTL allows for exploiting automata generated form LTL expressions [Montali et al., 2010a] and verification of participating individual services and whole services compositions, including a posteriori verification of properties and checking of service interaction. However, the LTL-based semantics of ConDec is not sufficient to present metric temporal constraints and although we note that the work of Westergaard and Maggi [2012] addresses this issue with a formal representation of time in DECLARE, we prefer to use a more expressive formal semantics such as Event Calculus. ConDec models can also be mapped to CLIMB/$\mathcal{S}$CIFF. This can represent activities, data elements, and time. Models expressed in CLIMB can be verified to ensure their correctness and consistency, as well as to check that their models meet the business goals and requirements. Tools such as the $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF proof procedures are able to reason about a complete execution trace, or upon a growing trace, by dynamically acquiring and processing occurring events as they happen.

At run time, the verification task aims at checking whether an evolving instance of the target system complies with all business constraints specified in the model. A more suitable formalism than CLIMB could be Event Calculus, for a number of reasons, including its simplicity and its compact representation, symmetry with respect to past and future, generality with respect to time orderings, executability and direct mapping to computational logic frameworks, modelling of concurrent events, immunity from the frame problem and explicit treatment of time and events [Kowalski and Sergot, 1989, Montali, 2010].

MSMAS model can potentially be represented and verified in many formalisms, each of them have its own degree of complexity and decidability. Figure 6-1 shows two possible methods to verify model designs using MSMAS Design tool by means of LTL automata reasonings, or CLIMB/$\mathcal{S}$CIFF formal proof procedures.

Although MSMAS supporting tool has the option to export MSMAS models as RDF/OWL, we chose not to rely on OWL reasoning because as noted by Fornara et al. [2013b], the Semantic Web technologies are not devised for modelling and monitoring dynamic systems due to two problems: the first relates to performing temporal reasoning, as OWL has no temporal operators, while the second relates to the monitoring of obligations with deadlines, as in these types of norms, it is important to reach a permanent state of fulfillment or violation once the deadline has elapsed. A further open problem in OWL-based reasoning is the matter of choosing which system components it is better – and possible – to represent in ontologies and reason about and which is more suitable for representation in an external application. The Semantic web standards offer no support or guidance on this.

In this chapter we show how MSMAS system norms can be mapped to CLIMB/$\mathcal{S}$CIFF to support static verification during design time, and how they can be mapped to EC to support monitoring at run time. We chose these particular technologies due to the availability of tools[1] that are ready to use and can be integrated and utilised for this purpose.

6.2 The CLIMB Framework and Language

The **C**omputational **L**ogic for the ver**I**fication and **M**odelling of **B**usiness constraints (CLIMB) [Montali, 2010] Language is part of the CLIMB framework proposed by Montali [2010]. The framework combines the constraint-based ConDec language with the Logic Programming-based $\mathcal{S}$CIFF framework to allow for static reasoning about ConDec based formal models as well as run-time and a-posteriori analysis of the execution traces. Figure 6-2 shows the logical architecture of the CLIMB framework.

The CLIMB language is a first-order logic-based language that is a subset of the $\mathcal{S}$CIFF language [Alberti et al., 2006], developed within the SOCS European Project[2] which aimed at

[1] MOBUCON and SCIFF Checker are available at (`https://www.inf.unibz.it/~montali/tools.html`, retrieved 31 Jan 2014)

[2] Societies Of ComputeeS (SOCS): a Computational Logic model for the description, analysis and verification of global and open societies of heterogeneous computees. IST-2001-32530, 2002-2005. Web-page: `http://`

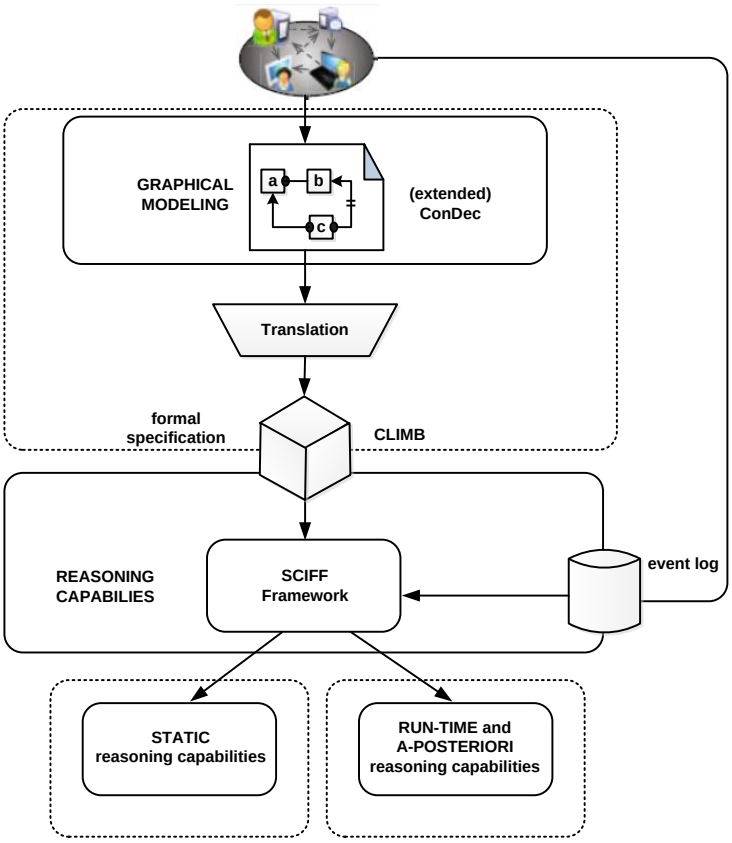


Figure 6-2: *The CLIMB Framework [Montali, 2010]*

establish a logic-based framework for the specification and verification of open and heterogeneous MAS.

CLIMB specifications include:

1. Specifications of system dynamics as a set of rules that links event occurrences with expectations about other event occurrences.
2. Specifications of the system's static aspects as a knowledge base.

The reason for choosing CLIMB over LTL, which is the original underlying formal presentation of ConDec, is that CLIMB can handle time and data in an explicit and quantitative way, which matches our aim to support real world applications that normally have time-dependent requirements such as e-commerce applications. In the following sections we cover the syntax of CLIMB language while more details on $\mathcal{S}$CIFF and CLP can be found in Appendix G.

6.2.1 Events in CLIMB

CLIMB is completely independent of the application domain: it has Events presented as terms, and it is up to the the developers to decide what events they want to track for modelling their application domain. For example, in MSMAS we consider the start of an activity, and the

`lia.deis.unibo.it/research/socs/`

completion of its execution as events. Since CLIMB is a first-order language, variables can be used inside events, this is an advantage that we use for modelling communication protocols by expressing time on the relations between the messages.

Each system instance can be expressed as a set of events occurring when the system entities act or interact with each other. CLIMB adopts an explicit, implicit and quantitative notion of time, where happened events are associated to a time variable. So an event Ev happened at a (discrete) time point *T* is denoted by the atom:

$$\mathbf{H}(Ev, T).$$

where Ev is a term and T is a numerical variable representing the time value depending on the chosen time structure by the application domain.

Time variables in CLIMB and CLP in general can be expressed qualitatively and quantitatively. For example:

- $T_2 > T_1$ is a qualitative time constraint that means T_2 is after T_1;
- $T \geq 5005412$ is a quantitative absolute value that specifies that T is after the absolute timestamp of 5005412;
- $T2 < T1 + 20$ is a relative time constraint that specifies that $T2$ is before $T1$ plus 20 time units.

The following events are sample events based on the custom communication protocol CCP_TranslateFile shown in Figure H-1 Chapter 4:

$$\begin{aligned} &send\_message((MSG\_RequestFile),\\ &\qquad (Supplier15, IR\_SupplierAgent),\\ &\qquad (TranslationService1, IR\_TranslationService),\\ &\qquad MSG11, \text{CCP\_TranslateFile}), 13334. \end{aligned}$$

$$\begin{aligned} &send\_message((MSG\_Refuse),\\ &\qquad (TranslationService1, IR\_TranslationService),\\ &\qquad (Supplier15, IR\_SupplierAgent),\\ &\qquad MSG55, \text{CCP\_TranslateFile}), 13344. \end{aligned}$$

where a system participant that plays IR_supplierAgent role, and is called $Supplier15$ sends a request message $MSG_RequestFile$ to another system participant that plays the IR_TranslationService role, and is called $TranslationService1$ with content $MSG11$ at time stamp 13334, then an inform message MSG_Refuse is sent from $TranslationService1$ to $Supplier15$ with content $MSG55$ after 10 time units past the first message at time stamp 13344.

CLIMB focuses on the dynamics of the system under study, hence it does not only address

"what" happened and "when", by using H atoms, but also, it offers an explicit presentation of "what" is expected (or not) to happen, and "when". Expectations are defined as abstract entities that capture the possible events that would make a system conform to its requirements. By definition the trace of the possible events is an evolution of the system towards its goals as a discrete time line where at each point of time, there is a set representing what is desired and what is not.

In CLIMB a positive expectation is used to express that an event Ev is expected to happen at a time T, and is expressed as:

$$\mathbf{E}(Ev, T)$$

while a negative expectation means that an event Ev is expected not to happen at time T, and is expressed as:

$$\mathbf{EN}(Ev, T)$$

The Ev term can be expressed with leaving some of its parts as variables, where for example parts can refer to events that have not yet occurred. This feature is very important in that it can help the system designer to include corrective actions in his system design that will be triggered automatically when an undesired event happens.

6.2.2 CLIMB Integrity Constraints

CLIMB social integrity constraints $\mathcal{IC}$s are a set of rules that specify in its simplest form, if an Ev_1 happened, then Ev_2 is expected to happen or not. Integrity constraints are presented as rules of the form:

$$Body \rightarrow Head.$$

The syntax for the *body* of each integrity constraint is a conjunction of BodyConjunct symbols, while the head is a disjunction of conjunctions of HeadConjunct symbols.

$$\bigwedge_i BConjunct_i \rightarrow \bigvee_j \left(\bigwedge_k HConjunct_{j,k} \right)$$

CLIMB defines $Body \rightarrow Head.$, "where $Body$ contains (a conjunction of) happened events, together with constraints on their variables, as well as Prolog predicates. And $Head$ contains (a disjunction of conjunctions of) positive and negative expectations, together with constraints and Prolog predicates, applied on their variables and/or on variables contained in the Body" [Montali, 2010]. Below is an example $\mathcal{IC}$, that states when the event SP starts achieving goal A, at time T_a, it is expected that SP will start to achieve goal B at time T_b:

$$\mathbf{H}(achieve\_goal(A, SP), T_A)$$
$$\rightarrow \mathbf{E}(achieve\_goal(B, SP), T_B) \wedge T_B > T_A.$$

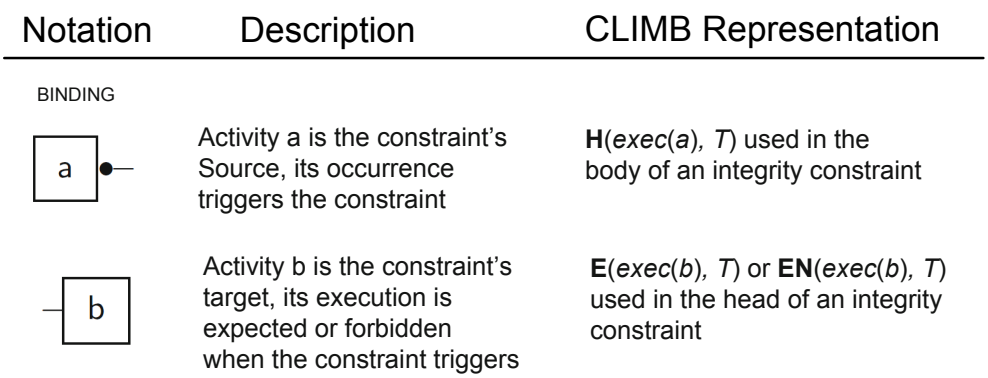


Figure 6-3: *Mapping ConDec relations to CLIMB [Montali, 2010]*

6.2.3 The Static Knowledge Base

Part of the system model that complements the dynamic aspects is the static background knowledge. These are the pieces of information that will not change during the execution, for example, information about the system components types such as the fact that a system goal is of the type Basic System Goal, or that an institutional role IR_1 is member of institution $Inst_B$. Explicit facts of this kind can be relevant during the evaluation of the system norms, where the system designer uses them to provide a definition to each predicate recalled in an $\mathcal{IC}$s. The store that holds all this pieces of knowledge is called Knowledge Base $\mathcal{KB}$ which is a standard logic program:

$$\mathcal{KB} ::= [Clause]^{\star}$$
$$Clause ::= Head \leftarrow Body$$
$$CBAtom ::= Literal$$
$$Literal ::= [\text{“not”}]Atom$$

For more details, refer to Appendix G, and [Kakas et al., 1993], [Alberti et al., 2008], and [Montali, 2010].

6.2.4 Expressing ConDec Relations in CLIMB

The use of ConDec's graphical notation has great advantages with regard to usability and user experience that suits general business developers, but additionally, these notations can be translated to an expressive formal language such as CLIMB to allow for formal verification and model checking. A ConDec model can be translated into a CLIMB specification by mapping its mandatory constraints onto CLIMB integrity constraints. Figure 6-3 shows how ConDec Notations are expressed in CLIMB, and in the remaining parts of this chapter we present the syntax of each defined system norm in MSMAS according to CLIMB.

One thing to observe is that only system goals, institutional roles, communication messages and activities connected to a mandatory constraint will be part of CLIMB model specifications,

while unconstrained system components do not appear at all. This approach preserves the openness feature that is needed for MASs, where any unconstrained activity can be executed an arbitrary number of times in any order without compromising the compliance of the running system with its design specifications.

6.3 MSMAS Norms Semantics in CLIMB

In MSMAS we assume that each activity is atomic meaning that it expected to happen or not at a specific point of time, hence it can be presented as a CLIMB term. We model each of these by means of a single happened event or expectation. Before presenting how each defined system norm is translated into CLIMB, we first define MSMAS events.

6.3.1 MSMAS Events

We have defined four types of norms as detailed in Chapter 4 , we list below the defined events related to each type of these system norms.

Events Related to System Goals Norms

In MSMAS we have defined six goal/goal relations as system norms to be used to constrain MSMAS system goals. To support the expression of these relations and link system goals to business activities, we define three event types associated with system goals. The syntax of expressing a MSMAS goal-related event in CLIMB is:

Happened goal-related event is expressed using CLIMB as:

$$H(GoalEventType(SG, SP), T).$$

where $GoalEventType$ is the type of goal-related event, SG is a system goal identifier, SP is a system participant identifier, and T is the time stamp. The defined list of goal-related event types are:

1. **achieve_goal:** This event marks the time when a system participant starts to act with intention to achieve a system goal.
   Happened $achieve\_goal$ **event is expressed using CLIMB as**:

   $$H(achieve\_goal(SG, SP), T).$$

   where $achieve\_goal$ is the event type, SG is a system goal, SP is a system participant, and T is the time stamp. An example of this event is: H ($achieve\_goal(SubmitUpdate$,

$SupplierAgent12$),12254) where a $SupplierAgent12$ starts to act to achieve the system goal $SubmitUpdate$ at the time stamp 12254.

2. **satisfy_goal:** This event happens to mark the completion of last activity needed to achieve a system goal.
   Happened $satisfy\_goal$ **event is expressed using CLIMB as**:

$$H(satisfy\_goal(SG, SP), T).$$

   where $satisfy\_goal$ is the event type, SG is a system goal, SP is a system participant, and T is the time stamp. An example of this event is: H ($satisfy\_goal$($SubmitUpdate$, $SupplierAgent12$),12265) where a $SupplierAgent12$ marks the successful completion of all activities were needed to achieve the system goal $SubmitUpdate$ at the time stamp 12265.

3. **drop_goal:** This event happens when a system participant stops its activities towards the achievement of a system goal for some reasons such as encountering an error, realising that the goal is not relevant any more, or changing plans.
   Happened $drop\_goal$ **event is expressed using CLIMB as**:

$$H(drop\_goal(SG, SP), T).$$

   where $sdrop\_goal$ is the event type, SG is a system goal, SP is a system participant, and T is the time stamp. An example of this event is: H ($drop\_goal$($SubmitUpdate$, $SupplierAgent12$),12259) where a $SupplierAgent12$ marks the unsuccessful completion of activities needed to achieve the system goal $SubmitUpdate$ at the time stamp 12259.
   Further Goal related events in support of Institutional Roles:
   The following events are defined to demonstrate how to utilise and link institutional roles' norms with system goals' norms, and to demonstrate the suitability of our approach for extension based on the application domain. We define two more goal events:
4. **delegate_goal:** This event happens when a system participant request from another system participant that plays an intuitional role under his authority to achieve a system goal on his behalf. This function can be used to facilitate cooperation or task delegation between system participants.
   Happened $delegate\_goal$ **event is expressed using CLIMB as**:

$$H(delegate\_goal((SPID_A, SPIR_A), (SPID_B, SPIR_B), SG), T).$$

where $delegate\_goal$ is the event type, $SPID_A$ and $SPIR_A$ are the ID and the institutional role of the system participant that delegates the goal described in the object SG, $SPID_B$, and $SPIR_B$ is the institutional role of the receiving system participant, and T is the time stamp.

5. **satisfyDelegated_goal:** This event happens when a system participant succeeds in achieving a goal that was delegated to it and reports back to the system participant that delegated to it this goal.
   Happened $satisfyDelegated\_goal$ **event is expressed using CLIMB as**:

$$H(satisfyDelegated\_goal((SP_A, IR_A), (SP_B, IR_B), SG), T).$$

All event types above can be observed as they happen, either when a system participant starts a business activity, completes a business activity, fails to complete a business activity or when the triggering condition of this event holds.

Events Related to System Institutional Roles Norms

To support the expression of role/role relations defined in MSMAS, we define two event types associated with institutional roles. The syntax of expressing a MSMAS goal-related event in CLIMB is:

Happened role-related event is expressed using CLIMB as:

$$H(RoleEventType(IR, SP), T).$$

Where $RoleEventType$ is the type of role-related event, IR is an institutional role, SP is a system participant, and T is the time stamp. The defined list of role-related event types are:

1. **play_role:** This event happens when a system participant intends to start playing an institutional role. Playing a particular institutional role enables the system participant to interact with other system participants in the behavioural pattern specified by this role.
   Happened $play\_role$ **event is expressed using CLIMB as**:

$$H(play\_role(IR, SP), T).$$

2. **drop_role:** This event happens when a system participant stops behaving according to the behavioural pattern specified by this role. If the system participant plays only one role in that institution, dropping this role means dropping the membership of this institution.

Dropping an institutional role might be for some reason such as changing of system participant goals or plans.
Happened $drop\_role$ **event is expressed using CLIMB as**:

$$H(drop\_role(IR, SP), T).$$

These event types can be observed as they happen either when a system participant starts a business activity linked with that role, or during a registration process, depending on system design and implementation.

Events Related to Communication Protocol Messages Norms

To support the expression of message/message relations defined in MSMAS, we define one event type associated with communication protocol messages.
Happened message-related event is expressed using CLIMB as:

$$H(MessageEventType((MID), (SID, SIR), (RID, RIR), MSG, CP), T).$$

where $MessageEventType$ is the type of message-related event, MID is the message id, SID is the id of the sender system participant and SIR is the institutional role it plays, RID is the id of the recipient system participant and RIR is the institutional role it plays, MSG is the content of the message, CP is the communication protocol the message is used within and T is the time stamp. The defined message-related event type is:

1. **send_message:** This event happens when a system participant sends a communication message to another system participant. Notice that the message is linked with the institutional role the system participant plays. **Happened** $send\_message$ **event is expressed using CLIMB as**:

$$H(send\_message(MID, (SID, SIR), (RID, RIR), MSG, CP), T).$$

This event type can be observed as it happens when a system participant sends a message to another.

Events Related to Business Activities

To support the expression of Activity/Activity relations and Activity constraints defined in MSMAS by means of ConDec^{++}, we define three event types associated with business activities. The syntax of expressing a MSMAS activity-related event in CLIMB is:

Happened activity-related event is expressed using CLIMB as:

$$H(ActivityEventType(AT, AID, (SPID, SPIR), BSGID), T).$$

where $ActivityEventType$ is the type of activity-related event, AT is activity type, AID is the activity id, $SPID$ is the id of the system participant that initiated the activity and $SPIR$ is the institutional role played by the system participant, $BSGID$ is the basic system goal id associated with the business process where this activity belongs and T is the time stamp. The defined activity-related event types are:

1. **start_activity:** This event happens when a system participant initiates the execution of an activity. **Happened** $start\_activity$ **event is expressed using CLIMB as**:

   $$H(start\_activity(AT, AID, (SPID, SPIR), BSGID), T).$$

   This event can be observed as it happens either when a system participant initiate an activity or sends a message to another system participant as part of a communication protocol that initiates the execution of a BPB by triggering the execution of the entry activity of a BPB.
2. **end_activity:** This event happens when a system participant successfully completes the execution of an activity, the successful execution means that the post conditions of the business activity hold. **Happened** $end\_activity$ **event is expressed using CLIMB as**:

   $$H(end\_activity(AT, AID, (SPID, SPIR), BSGID), T).$$

   This event can be observed as it happens either when a system participant completes the execution of an activity and its post conditions hold, or if the system participant obtains new knowledge confirming that this activity's post conditions hold. This scenario can happen in the case of system participants with hierarchal roles relation, where the child role reports the results of the execution of a task that was delegated to it by the parent role.
3. **terminate_activity:** This event happens when a system participant does not complete the execution of an activity, or ends the execution unsuccessfully, which means that part or all of the post conditions of the business activity do not hold. **Happened** $terminate\_activity$ **event is expressed using CLIMB as**:

   $$H(terminate\_activity(AT, AID, (SPID, SPIR), BSGID), T).$$

   This event can be observed as it happens either when a system participant stops the execution of an activity for some reason, such as encountering an error, or it drops the goal that required the execution of this activity and the activity becomes irrelevant.

6.3.2 Goal/Goal Relation Formalisation in CLIMB

Given a MSMAS model, each Goal/Goal relation present in the model is captured as a corresponding fact of the type:

$goal\_goal\_relation(A, B, Type)$

where A and B are system goals and $Type$ is the type of goal/goal relation. For example, $goal\_goal\_relation(BSG\_Collect\_User\_Data, BSG\_Enrol\_User, sequential\_goals)$, expresses that a precedence/sequential goals relation holds between $BSG\_Collect\_User\_Data$ and BSG_Enrol_User goals. All these facts are grouped together inside a "System Goals Norms" knowledge base $\mathcal{KB}_{sgn}$.

The goal/goal relations described in Chapter 4 in Figure 4-18 Page 112 are then formalised by means of $\mathcal{IC}_s$ that follow the ConDec^{++} to CLIMB translation presented in [Montali, 2010]. Such constraints are grouped together into an integrity constraint set $\mathcal{IC}_{sgn}$.

Sequential Goals (SG):

$$
\begin{aligned}
&goal\_goal\_relation(A, B, sequential\_goals) \\
&\wedge \mathbf{H}(achieve\_goal(A, SP), T_A) \\
&\quad \rightarrow \mathbf{E}(achieve\_goal(B, SP), T_B) \wedge T_A < T_B.
\end{aligned}
$$

Notice that the constraint is instantiated for every goal/goal relation of type $sequential\_goals$ contained into $\mathcal{KB}_{msmas}$.

Sequential Goals (SG) with Time Constraint (n, m):

$$
\begin{aligned}
&goal\_goal\_relation(A, B, sequential\_goals, (n, m)) \\
&\wedge \mathbf{H}(achieve\_goal(A, SP), T_A) \\
&\quad \rightarrow \mathbf{E}(achieve\_goal(B, SP), T_B) \wedge T_B => T_A + n \wedge T_B <= T_A + m.
\end{aligned}
$$

Joint Goals (JG):

$$
\begin{aligned}
&goal\_goal\_relation(A, B, joint\_goals) \\
&\wedge \mathbf{H}(achieve\_goal(A, SP), T_A) \\
&\quad \rightarrow \mathbf{E}(achieve\_goal(B, SP), T_B) \wedge T_B > T_A. \\
&goal\_goal\_relation(A, B, joint\_goals) \\
&\wedge \mathbf{H}(achieve\_goal(B, SP), T_B) \\
&\quad \rightarrow \mathbf{E}(achieve\_goal(A, SP), T_A) \wedge T_A < T_B.
\end{aligned}
$$

Joint Goals (JG) with Time Constraint (n, m):

$$
\begin{aligned}
&goal\_goal\_relation(A, B, joint\_goals, (n, m)) \\
&\wedge \mathbf{H}(achieve\_goal(A, SP), T_A) \\
&\quad \rightarrow \mathbf{E}(achieve\_goal(B, SP), T_B) \wedge T_B > T_A + n \wedge T_B <= T_A + m. \\
&goal\_goal\_relation(A, B, joint\_goals) \\
&\wedge \mathbf{H}(achieve\_goal(B, SP), T_B) \\
&\quad \rightarrow \mathbf{E}(achieve\_goal(A, SP), T_A) \wedge T_B > T_A + n \wedge T_B <= T_A + m.
\end{aligned}
$$

Notice that the constraint is instantiated for every goal/goal relation of type $sequential\_goals$ contained in $\mathcal{KB}_{msmas}$.

Coupled Goals (CF):

$$
\begin{aligned}
&goal\_goal\_relation(A, B, coupled\_goals) \\
&\wedge \mathbf{H}(achieve\_goal(A, SP), T_A) \\
&\quad \rightarrow \mathbf{E}(achieve\_goal(B, SP), T_B). \\
&goal\_goal\_relation(A, B, coupled\_goals) \\
&\wedge \mathbf{H}(achieve\_goal(B, SP), T_B) \\
&\quad \rightarrow \mathbf{E}(achieve\_goal(A, SP), T_A).
\end{aligned}
$$

Notice that the constraint is instantiated for every goal/goal relation of type $coupled\_goals$ contained in $\mathcal{KB}_{msmas}$.

Disjoint Goals (DF):

$$
\begin{aligned}
&goal\_goal\_relation(A, B, disjoint\_goals) \\
&\wedge \mathbf{H}(achieve\_goal(A, SP), T_A) \\
&\quad \rightarrow \mathbf{EN}(achieve\_goal(B, SP), T_B). \\
&goal\_goal\_relation(A, B, disjoint\_goals) \\
&\wedge \mathbf{H}(achieve\_goal(B, SP), T_B) \\
&\quad \rightarrow \mathbf{EN}(achieve\_goal(A, SP), T_A).
\end{aligned}
$$

Notice that the constraint is instantiated for every goal/goal relation of type $disjoint\_goals$ contained in $\mathcal{KB}_{msmas}$.

The remaining relation is the Child/Parent Goals that states that one goal is a subgoal of another. In the following $\mathcal{IC}$ we express the idea that when any system participant starting to achieve a system goal, it effectively starts to achieve this goal's super goal.

ChildParent Goals (CF):

$$goal\_goal\_relation(A, B, childParent\_goals)$$
$$\wedge\, \mathbf{H}(achieve\_goal(A, SP), T_A) \wedge$$
$$\rightarrow \mathbf{E}(achieve\_goal(B, SP), T_B).$$

Notice that the constraint is instantiated for every goal/goal relation of type $childParent\_goals$ contained into $\mathcal{KB}_{msmas}$.

6.4 Static Verification of MSMAS Models

Formal modelling methods of software comprise two activities: *formal specification* and *verification*. The formal specification permits the definition of a precise specification of the system behaviour and its components. While the verification aims at proving that the system model design complies with the intended requirements, and meets the desired properties. In this section we focus on the static verification of MSMAS models during design time using $\mathcal{S}$CIFF proof procedure.

6.4.1 Compliance in $\mathcal{S}$CIFF

$\mathcal{S}$CIFF which is based on Abductive Logic Programming (ALP), uses declarative semantics to capture the meaning of expectations. In particular, $\mathcal{S}$CIFF declaratively captures the notion of *compliance* of a system execution trace with the modelled specification by considering positive and negative expectations as abducible predicates, and by introducing the notion of *fulfillment*. Starting from the knowledge base and the set of happened events contained in the analyzed trace of the system (which extends the knowledge base with information about the dynamics), expectations are hypothesized consistently with the ICs and with an expectation-consistency rule stating that no event can be expected to happen and not to happen at the same time. A positive (respectively negative) expectation is then judged as fulfilled (respectively violated) if there exists a corresponding matching happened event in the trace. This can be considered as a sort of *hypothesis confirmation* step, where the hypothesized courses of execution match with an actual behaviour.

To deal with the possibility that only a partial trace of the system is known, $\mathcal{S}$CIFF can maintain the generated expectations *pending*, awaiting further happened events to judge their fulfillment. A special case is when it is known that no further happened event will ever occur. In this case, all pending positive expectations are considered violated, whereas all pending negative expectations are considered fulfilled: no matching event will ever be found in the future.

This declarative notion of compliance has an operational counterpart in the $\mathcal{S}$CIFF proof

procedure, which concretely realises an inference mechanism to (i) dynamically acquire happened events, reporting about the evolution of the system dynamics, (ii) use the modelled knowledge base and integrity constraints so as to generate expectations about the courses of execution, and (iii) match expectations with happened events, deciding their fulfillment. Execution traces which fulfill all the generated expectations are then deemed *compliant* with the specification.

An extension of the $\mathcal{S}$CIFF proof procedure, called g-$\mathcal{S}$CIFF, can be used to prove properties of the model at design time, i.e., without having an explicit trace of the system Montali et al. [2010b]. Given a property, g-$\mathcal{S}$CIFF tries to generate a (partially specified) trace showing that the property can be satisfied while respecting all the modelled ICs. Intuitively, this is done by transforming every pending positive expectation into a corresponding happened event and checking that no negative expectation is violated.

Finally, we observe that while termination of the proof procedures cannot be guaranteed in general, all the techniques developed to check termination of (abductive) logic programs can be seamlessly applied to $\mathcal{S}$CIFF.

By putting together the translation principles presented above, we obtain a full CLIMB/$\mathcal{S}$CIFF specification constructed as follows:

$$\mathcal{P}_{msmas} \equiv \langle \mathcal{KB}_{msmas}, \{\mathbf{E}/2, \mathbf{EN}/2\}, \mathcal{IC}_{msmas} \rangle$$

$$\text{where} \quad \mathcal{KB}_{msmas} \triangleq \mathcal{KB}_{sgn} \cup \mathcal{KB}_{irn} \cup \mathcal{KB}_{cpn} \cup \mathcal{KB}_{act}$$

$$\text{and} \quad \mathcal{IC}_{msmas} \triangleq \mathcal{IC}_{sgn} \cup \mathcal{IC}_{irn} \cup \mathcal{IC}_{cpn} \cup \mathcal{IC}_{act}$$

The $\mathcal{S}$CIFF and g-$\mathcal{SCIFF}$ proof procedures can consequently be applied to reason about MSMAS models. Two approaches can be used as highlighted in Figure 6-1: the first one through the $\mathcal{S}$CIFF proof procedure that can be used to check compliance of simulated event traces, while the second approach through g-$\mathcal{S}$CIFF that can be applied to verify whether the MSMAS model of the system meets some desired properties.

Given a set **HAP** of happened events, $\mathcal{P}_{msmas}$ leads to formulate an abductive set **EXP** that contain positive and negative expectations, which reflects the events that are expected (or not) to occur in the state of affairs obtained after the execution of the events in **HAP**. In this respect, we take advantage of the declarative semantics of $\mathcal{S}$CIFF to tackle three basic reasoning tasks: *consistency*, *fulfillment*, and *conformance*.

Consistency states that a MSMAS event cannot be expected to happen and expected not to happen at the same time. Technically, for each (ground) MSMAS event $Event$ and timestamp T, consistency requires that $\{\mathbf{E}(Event, T), \mathbf{EN}(Event, T)\} \not\subseteq \mathbf{EXP}$. Notice that consistency is not checked by the proof procedures by effectively grounding the expectations, but by using variables and CLP constraints to maintain an intensional, "symbolic" presentation of (classes of) expectations, using constraint-solving to detect clashes between positive and negative expectations.

Fulfillment expresses the semantics of expectations. In particular, we say that a positive expectation $\mathbf{E}(Event, T) \in \mathbf{EXP}$ is fulfilled by a set **HAP** of happened events if and only

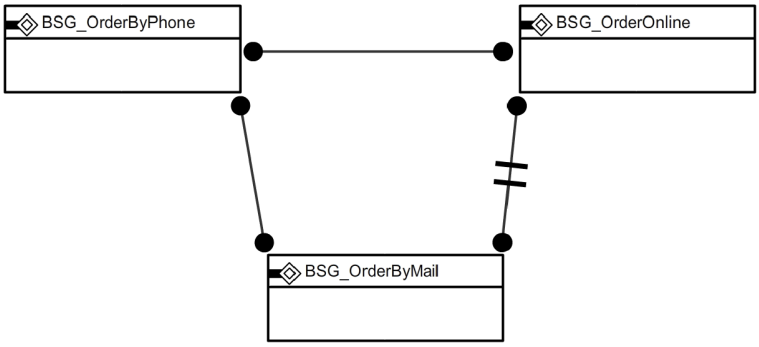


Figure 6-4: *Example Composite Business Process Model with Goal/Goal Norms*

if $\mathbf{H}(Event, T) \in$ **HAP**, i.e., a corresponding happened event has occurred. Specifically, a negative expectation $\mathbf{EN}(Event, T) \in$ **EXP** is fulfilled by a set **HAP** of happened events if $\mathbf{H}(Event, T) \notin$ **HAP**, i.e., no corresponding happened event has occurred. Furthermore, we say that **EXP** is fulfilled by **HAP** if every expectation in **EXP** is fulfilled by **HAP**.

Conformance combines the notion of consistency and fulfillment to characterize whether a trace of the system respects all the constraints imposed by the MSMAS model. In particular, given a goal G and a complete trace of the system **HAP**, we say that **HAP** *conforms to* the MSMAS model satisfying G if:

$$\mathcal{KB}_{msmas} \cup \mathbf{HAP} \cup \mathbf{EXP} \models G$$
$$\mathcal{KB}_{msmas} \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC}_{msmas}$$
$$\mathbf{EXP} \text{ is consistent}$$
$$\mathbf{EXP} \text{ is fulfilled by } \mathbf{HAP}$$

6.4.2 Verification Example

Consider the composite business process model shown in Figure 6-4. The model contains three goal/goal relations set as system goal norms

Looking closely, we observe that the specifications of $BSG\_OrderByPhone$, $BSG\_OrderByMail$ and $BSG\_OrderOnline$ eventually lead to an inconsistency, as soon as an agent starts to achieve one of these system goals: $BSG\_OrderByPhone$ and $BSG\_OrderOnline$ are joint goals, and so are $BSG\_OrderByPhone$ and $BSG\_OrderByMail$, which implies that also $BSG\_OrderOnline$ and $BSG\_OrderByMail$ have to be joint goals, while the model constrains them to be disjoint. This inconsistency can be detected by the $\mathcal{S}$CIFF proof procedure when a partial trace of the system is analyzed. With g-$\mathcal{S}$CIFF, instead, the problem can, e.g., be detected when the modeller poses the following query: *is it possible for an agent to achieve the goal of $BSG\_OrderByMail$?* Observe that a negative answer to this query would seriously question the correctness of the MSMAS model, because it would attest that

$BSG\_OrderByMail$ is always an "empty" goal in any possible execution.

The previous sections showed how we can use $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF for static verification of the designed MSMAS models. The following sections cover continuous checking or monitoring of the system norms during execution.

Although $\mathcal{S}$CIFF can be used for reactive checking it has serious issue that makes it unsuitable for monitoring, namely that when a violation is detected by $\mathcal{S}$CIFF, the proof procedure immediately terminates the computation, returning a negative answer. This could be sufficient for the system designer interested in only checking compliance, but the core principle of self-managed systems is for the system to be able to monitor itself and to apply corrective actions. In this sense the detection of violations by means of monitoring is part of a feedback loop and the other parts of self-management include the management of violations as part of the interaction.

In self-managing systems when a violation is detected through means of continuous monitoring, it is important to allow the consequent exceptional events to happen and continue the monitoring process to check that the interacting entities are correctly reacting to that violation. We utilise a formal method for online monitoring as inspired by the work of Montali et al. [2011] using Event Calculus (EC) and Logic Programming. We start by presenting a summary of EC, how ConDec can be formalised in EC, then we present in detail our approach to express MSMAS Norms in EC and finally we show how a monitoring system can be used to detect violations of MSMAS system norms.

6.5 Event Calculus Framework

Kowalski and Sergot [1986] proposed Event Calculus (EC) as a general framework to reason about time dependent properties called $fluents$[3], and $events$ that affect these fluents over time. Fluents are relations whose truth value may vary from time point to another and in EC, semantically, the universe is partitioned into disjoint sub-universes. A variable ranges over its own sub-universe, and a term will denote an element in its corresponding sub-universe. While the original Event Calculus OEC was proposed by Kowalski and Sergot [1989] other versions have followed such as the Simplified Event Calculus SEC that was proposed by Kowalski [1992], the Basic Event Calculus BEC [Mueller, 2004], and the Discrete Event Calculus DEC [Mueller and Sutcliffe, 2005].

EC offers a logical way to present time and to provide a solid theoretical basis for reasoning about complex requirements in event based Systems. In the Event Calculus the time is explicitly presented making it possible to express both qualitative and quantitative time constraints and being based on first-order logic it provides great expressiveness.

One other advantage of EC is that it can be completely axiomatised by relying on the

[3]The name is inspired by Newton's treatise on calculus, where the assumption is that all variables are implicitly dependent on time.

$happens(Ev, T)$	Event Ev happens at time T
$hold\_at(F, T)$	Fluent F holds at time T
$hold\_for(F, [T_1, T_2])$	Fluent F holds during the interval $[T_1, T_2]$
$initially(F)$	Fluent F holds in the initial state of the system
$initiates(Ev, F, T)$	Event Ev initiates Fluent F at time T
$terminates(Ev, F, T)$	Event Ev terminates Fluent F at time T

Table 6.1: *The Event Calculus Ontology*

Horn clause[4] subset of classical logic and complemented with Negation As Failure [5] [Montali, 2010]. This means the formalisation in EC can be directly executed as a logic program.

The general theory axiomatising EC that defines the meaning of the predicates supported by the calculus is called the EC **Ontology** and it contains the set of predicates shown in Table 6.1. An EC theory is a way to formalise how domain-specific events affect the domain-specific fluents, which can then be constituted as a logic program whose clauses define the system initial state and relate events occurrences with fluents initiation and termination Montali et al. [2011]. In the following subsections we give more details about Event Calculus ontology and theories.

6.5.1 The Event Calculus Ontology

Shanahan [1999] describes EC as "a logical mechanism that infers what is true when, given what happens when and what actions do." According to this description the main three concepts of EC are: (i) an $event$ (ii) that happens at a point of $time$, and (iii) and one or more properties called $fluents$ are affected over time as events occur. An EC specification is constituted by two theories:

- EC **Ontology**: a domain-independent general theory axiomatising the meaning of the predicates supported by the calculus.
- **Fluents**: the system specifications as a domain theory that exploits the predicates of the EC ontology.

The capability of an event Ev to make a fluent F true at time T is formalised by stating that the event initiates the fluent, while this relationship is formalised by stating that the event terminates the fluent to express that it makes it false.

The execution trace is defined as a set of occurred events characterising fully or partially an instance of the system under study. Each event in the execution trace is considered an atom (one event at a point of time). The combination of the domain knowledge and an execution trace leads to infer "what is true when".

[4] Horn clause is a disjunction of literals with at most one positive literal, and is named after the logician Alfred Horn

[5] Negation As Failure (NAF) is an inference rule in logic programming, used to derive $\mathrm{not}p$ from failure to derive p

6.5.2 The Event Calculus Theories

An EC theory is a way to formalise how domain-specific events affect domain-specific fluents. It is constituted by a logic program whose clauses define the system initial state and relates events occurrence with fluents initiation and termination. EC theory may also provide a set of conditions that have to hold to declip or clip a fluent. In EC theory variables are universally quantified within the scope of the entire clause. Hence, the following statement $initiates(Ev; F; T)$ states that event Ev causes F to hold at every time given that F is not already holding; otherwise Ev has no effect.

Example: consider a system characterised by two events, a payment event expressed as $pay(orderId, amount)$ that causes a variable called $OrderStatus$ to have the value $Billed$, and shipping event expressed as $ship(orderId)$ that causes another variable $DespatchStatus$ to be have the value $Shipped$. The $ship$ event is triggered when the $OrderStatus$ gets the value $Billed$. We would like to infer the status of each instance of dispatch either started or completed at any point of time. This can be modelled in the EC by introducing two multi-valued fluents $OrderStatus$ and $DispatchStatus$ where the current status of each of them corresponds to a system state, we can then use the following EC theory to relate them to the payment event:

$$Initially(OrderStatus(Started))$$
$$Initially(DespatchStatus(Started))$$

$$terminates(pay(id, a), OrderStatus(X), T).$$
$$terminates(ship(id), DispatchStatus(X), T).$$

$$initiates(pay(id, a), OrderStatus(X), T) \longleftarrow$$
$$holds\_at(OrderStatus(X), T)$$
$$\wedge\, initiates(ship(id), T).$$

$$initiates(ship(id), DispatchStatus(X), T) \longleftarrow$$
$$holds\_at(DispatchStatus(X), T).$$

The first clause models that the value of $OrderStatus$ is initially $Started$. The following clauses state that when a payment event occurs, the current value of $OrderStatus$ ceases to hold, and when a ship event occurs, the current value of $DespatchStatus$ ceases to hold. The following clauses updates the value $OrderStatus$ by initiating a new fluent whose value is $Billed$ and trigger the ship event.

6.5.3 Reasoning About EC Theories

There are three reasoning tasks that can be done in the EC setting:

- **Abductive Reasoning**: this task starts from an EC domain theory and a query presenting a desired state and it tries to generate a sample trace which achieves that state and respects the domain theory. The synthesised trace is often considered as a plan.
- **Deductive Reasoning**: this task takes an external trace and an EC domain theory to infer the validity intervals of fluents and answer to given queries. Deductive reasoning is done either starting from the query and reasoning backward.
- **Inductive Reasoning**: this task takes an external trace and the validity intervals of fluents and it tries to generalise the connection between fluents to produce general theory.

Example: Let us consider the EC theory described in the previous section and a specific stream of events. At the beginning of the execution, CEC infers that the value of $OrderStatus$ is $Started$ from time point 0 to an unknown future time point. Now suppose that a $pay(id, a)$ event occurs at time 5, According to the EC theory, the new value of $OrderStatus$ changes to $Billed$ and the event $ship(id)$ is triggered. The initiation of event $ship(id)$ causes the $DispatchStatus$ to change from its initial value $Started$ to $Shipped$.

Our formalisation of MSMAS system norms is inspired by [Montali et al., 2011], however we model the transitions of the state of the system norms as an effect of various events affecting the state of each activity. In our approach each system norm is considered as a non-atomic activity and it follows the system norm lifecycle shown in Figure 6-7. The following section explains this in more detail.

6.6 MSMAS Norms Semantics in Event Calculus

In MSMAS, achieving a system goal or playing an institutional role is considered a non-atomic durative activity where the activity execution spans a time period and is motivated by multiple event occurrences. Defining these activities requires three steps as stated by Montali et al. [2011]:

- the identification of the atomic events characterising the execution of a MSMAS non-atomic activity.
- the definition of the activity life cycle that describes the acceptable orderings of such events.
- an extension of the graphical notation to handle properly the non-atomic activities.

In the following subsections we define the triggering events and the life cycle of MSMAS and present the formalisation of them and MSMAS norms in Event Calculus.

6.6.1 MSMAS Activities Life cycle and State Transitions Triggering Events

The system execution causes the SNs to evolve over time, due to the occurrence of events. Thus, in MSMAS the execution of business processes to achieve a system goal is a collection of atomic activities and We segregate MSMAS sets of events which affect the system norms into two groups: (i) system goals (SG) and business activities (BA) events, and (ii) institutional roles (IR) and communication messages (CM) events.

Inspired by the work of Montali et al. [2011] who models the activity life cycle as a collection of four states: $active$, $error$, $completed$, and $cancelled$, we define a slightly simpler life cycle of only three states: $active$ (s)[6], $inactive$ (n)[7], and $complete$ (c)[8] (Figure 6-5(a)) that governs system goals and business activities-related actions. We also define a yet simpler life cycle of only two states: $active$, and $inactive$ (Figure 6-5(b)) that governs institutional roles and communication messages-related actions. Our SNs life cycle have fewer states than those defined in [Montali et al., 2011] because some of MSMAS activities are binary: consider Institutional Roles, for example, where a system participant either plays the role or not. The system execution events implicitly manipulate each non-atomic activity instance causing it to change state. Each activity instance has various states that can be presented as a state machine where the possible state transitions are called MSMAS non-atomic activity life cycles. When an event causes an instance to start, it causes the state of this particular instance to change to $active$. MSMAS SG and BA related events do not allow for further transitions once the activity instance reaches the $complete$ state, while MSMAS IR and CM related events do not have $complete$.

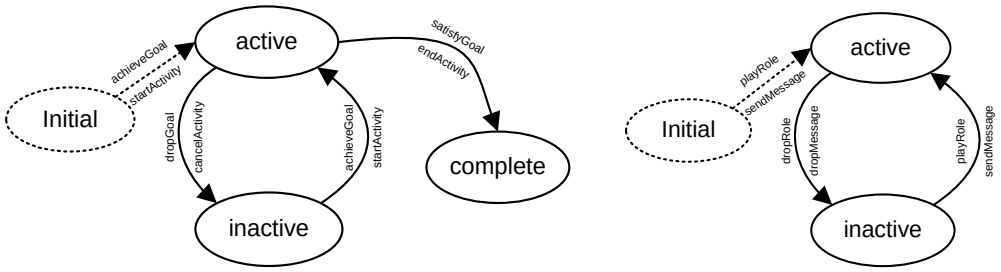
Triggering events are those events that cause the activity instance to move from one state to another. For example the system goals activity triggering events are expressed as:

$$Event(GoalEventType(SGID, SPID), T).$$

where $GoalEventType$ can be: $achieve$, $satisfy$, or $drop$ goal. $SGID$ is a system goal ID, $SPID$ is the ID of the system participant, and T is the time stamp of the event.

The non-atomic activity instance can be in only one state at any time and each activity group has a specific set of states and a different set of state transitions. Such state transitions are triggered by events executed by the system participants. The system designer, through their design, implicitly defines which events trigger and motivate the state transitions of each activity. For example, consider a set of events related to one the system goals shown in Figure 6-6: the state of $BSG\_PublishSupplierStock$ system goal changes as a result of events such as the event
$Ev(achieve\_goal(BSG\_PublishSupplierStock, SupplierAgnet), 1254)$ which causes this activity instance to have the state $active$. The expected next event could be either $satisfy\_goal$ or $drop\_goal$. If the next event is

[6] started
[7] not started/cancelled
[8] completed



(a) SG and BA Life Cycle (b) IR and CM Life Cycle

Figure 6-5: *MSMAS Events and Goals, Activities, Roles, Messages Generic Life Cycle Modelled as Non-atomic Activities*

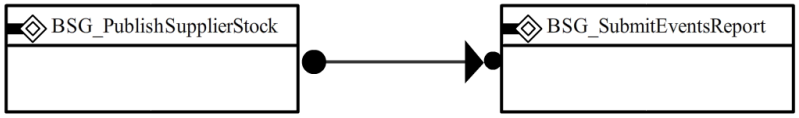


Figure 6-6: *A Composite Business Process Model (CBC_MaintainSupplierStock) with two Basic System Goals and one Goal/Goal Norm*

$Ev(satisfy\_goal(BSG\_PublishSupplierStock, SupplierAgnet), 1262)$ then the activity instance reaches the state $complete$ where no further transitions are expected. The full list of triggering events is:

System Goals:	$(achieve - satisfy - drop)$
Business Activities:	$(start - complete - cancel)$
Institutional Roles:	$(playRole - dropRole)$
Communication Messages:	$(sendMessage - cancelMessage)$

We express MSMAS norms by two different theories: (i) a domain-independent theory, that formalises the life cycle of time-aware system norms (Figures 6-5(a), and 6-5(b)), and gives semantics to the state transitions by relating the initiation/termination of norm-related fluents, and (ii) a theory that describes a specific domain, and defines the relation between the domain-specific events and fluents/norms.

6.6.2 Domain-independent Theory of MSMAS Activity Life Cycles

According to the required steps mentioned at the end of Section 6.6 Page 207, we now define formally the activity life cycles. When an event occurs it may affect MSMAS activities and cause the system to move from its current state to a new state according to the activity life cycles in Figures 6-5(a) and 6-5(b). We follow the same formalisation technique proposed by Montali et al. [2011] to make use of currently available tools that support monitoring event

based systems[9].

We use the fluent $ai\_state(i(id, a), s)$ to denote that an activity instance identified by i of system component a is currently in state s.

Axiom 1 (Effective Start). *An activity instance is effectively $started$ (s) by a start event occurrence as long as it is not already started. The opposite state $\neg started$ is defined in Axiom 3*:

$$happens(start(ID, A), T) \leftarrow happens(ev(ID, s, A), T)$$
$$\wedge\neg initiates(ev(ID, s, A), ai\_state(i(ID, A), active), T).$$

The effective start triggers a creation of the corresponding activity instance, transferring the identifier and placing the instance in the active state:

$$initiates(start(ID, A), ai\_state(i(ID, A), active), T).$$

Axiom 2 (Effective Completion). *An activity instance with name A and identifier ID is effectively completed (c) at time T if a completion event matching A and ID occurs at some time T, such that the activity instance is $active$ at time T*:

$$happens(complete(ID, A), T) \leftarrow happens(execute(ID, c, A), T)$$
$$\wedge holds\_at(ai\_state(i(ID, A), active), T).$$

Effective completion triggers transition to the completed state:

$$terminates(complete(ID, A), ai\_state(i(ID, A), active), T).$$
$$initiates(complete(ID, A), ai\_state(i(ID, A), complete), T).$$

Axiom 3 (Effective Cancellation). *The effective cancellation n of an activity instance mirrors the axioms used for effective start as its semantics is $\neg started$*:

$$happens(cancel(ID, A), T) \leftarrow happens(execute(ID, n, A), T)$$
$$\wedge holds\_at(ai\_state(i(ID, A), active), T)$$
$$\wedge\neg initiates(ev(ID, n, A), ai\_state(i(ID, A), inactive), T).$$

Effective Cancellation triggers transition to the inactive state:

$$terminates(cancel(ID, A), ai\_state(i(ID, A), active), T).$$
$$initiates(cancel(ID, A), ai\_state(i(ID, A), inactive), T).$$

6.6.3 MSMAS Norm Instances and their States

During execution, the triggering events defined earlier not only have impact on their associated activity states but also on each system norm associated with such activities. For example let us reconsider the $joint\_goals$ SG norm between $BSG\_PublishSupplierStock$ and $BSG\_SubmitEventsReport$ system goals as shown in Figure 6-6, where $BSG\_PublishSupplierStock$ is the source and $BSG\_SubmitEventsReport$ is the target goal. This norm is triggered every time a system participant successfully satisfies/achieves this goal, that is, the goal activity instance reaches the state $complete$, expecting the completion of the target goal $BSG\_SubmitEventsReport$

[9] A number of tools including jREC reasoner available via `https://www.inf.unibz.it/~montali/tools.html`, retrieved 31 Jan 2014

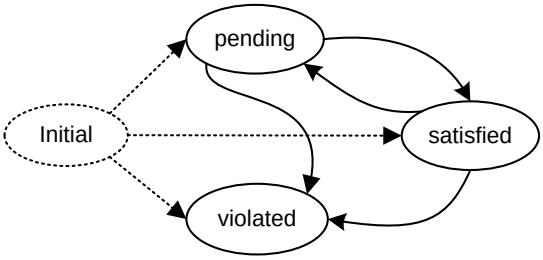


Figure 6-7: *MSMAS System Norms Generic Life Cycle Modelled as Non-atomic Activities*

at some point in the future. To capture this behaviour, every execution that satisfies the source goal creates a fresh instantiation of the norm and this particular norm instance is placed in a $pending$ state, waiting for the occurrence of the target goal.

Each system norm instance represents the application of the system norm in a particular context.

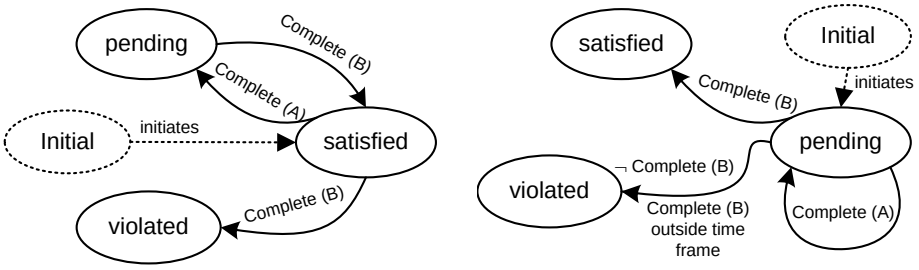
In this work, we present only MSMAS System Goals norms and MSMAS Institutional Roles Norms as examples to show the state transitions of MSMAS norms as an effect of the triggering activities as defined in the life cycles shown in Figures 6-5(a), and 6-5(b). The remainder of MSMAS norms follow these examples.

In our representation, we associate a unique identifier to each modelled norm, the term $i(id,$ $a,$ $t)$ denotes the instance of norm id has been created by the execution of a at time t. Each norm instance can be in only one state at any point of time and it obeys the generic life cycle of MSMAS norms as shown in Figure 6-7, where $pending$ is a transient state meaning that the SN is waiting for the occurrence of some event, $satisfied$ is either a transient or a permanent state indicating that the execution trace is currently compliant with the SN, and $violated$ is a permanent state indicating that the instance has been violated.

We employ a multi-valued fluent $sn\_state(I,$ $S)$ to present the state of each system norm instance, where I is the SN instance and S is the current state of I which can be one of the possible three values ($pending$ - $satisfied$ - $violated$). For example, $sn\_state(i(id,$ $a,$ $t),$ $satisfied)$ represents the fact that SN instance $i(id,$ $a,$ $t)$ is currently satisfied.

6.6.4 Formalising MSMAS Norms as a Domain-Specific Theory

Now we present our axiomatisation of MSMAS SNs as an EC domain-specific theory. We need both domain-specific theories with the domain-independent theories defined in section 6.6.2 to reason about the execution traces and to verify if the system complies/violates its specified requirement at run time. Our formalism depends on the creation of – and the state transitions of – SNs instances, as a result of event occurrences. To support the state manipulation we use the following five predicates defined by Montali et al. [2011], where the first two rules deal with the creation of a SN instance and setting its state. The third predicate is used to check the current state of a SN instance and the last two rules capture the state transitions.



(a) SN I Life Cycle (b) SN II Life Cycle

Figure 6-8: *MSMAS System Norms Life Cycle Modelled as Non-atomic Activities*

$initially(sn\_state(I,S)) \leftarrow init\_state(I,S).$
$initiates(E, sn\_state(I,S), T) \leftarrow creation(E,T,I,S).$

$cur\_state(I,S,T) \leftarrow holds\_at(sn\_state(I,S),T).$

$terminates(E, sn\_state(I,S_1), T) \leftarrow trans(E,T,I,S_1,S_2).$
$initiates(E, sn\_state(I,S_2), T) \leftarrow trans(E,T,I,S_1,S_2) \wedge holds\_at(sn\_state(I,S_1),T).$

Now let us consider two MSMAS norms to be expressed formally in terms of EC theories. Figure 6-8(a) and Figure 6-8(b) show the life cycle of these two SNs. For more formalisation of MSMAS norms in EC refer to Appendix I. The first example (SN I) is to illustrate the formalism of a joint goals relation, while the second example (SN II) is to illustrate how a time-constrained norm can be formally represented. A list of more norms and their formalism in EC can be found in Appendix I.

SN I: Joint Goals System Norm:

A Joint Goals relation between two system goals (source: $goal_A$ - target: $goal_B$) states that $goal_B$ must be achieved after successfully achieving $goal_A$, and $goal_A$ must be successfully achieved before achieving $goal_B$. This means the initial state of such a SN should be set to $satisfied$ (*Axiom 4*) then within an execution trace for each instance that completes successfully $goal_A$ the fluent representing this system norm instance should move to $pending$ state (*Axiom 5*), waiting for a complete event that successfully completes $goal_B$ to move to state $satisfied$ (*Axiom 6*). Two cases lead to a violation of the SN: (i) if it is in its initial state $satisfied$ then an event of successful completion of $goal_B$ occurs (*Axiom 7*) or (ii) when it is pending, and a case complete event is received that announces that no further events to happen (*Axiom 8*). The following axioms model these different cases:

Axiom 4 (Joint Goals Creation). *Each joint goals system norm is associated to a unique instance, created and put in the satisfied state when the case is started*:

$$init\_state(i(id, start, 0), sat).$$

Axiom 5 (Joint Goals Pending). *A satisfied joint goals SN instance becomes* $pending$ *when its source system goal is completed*:

$$trans(complete(-, A), T, i(id, A, T_i), sat, pend) \leftarrow$$
$$cur\_state(i(id, A, T_i), sat, T) \wedge T \geq T_i.$$

Axiom 6 (Joint Goals Fulfilment). *A pending joint goals SN instance becomes* $satisfied$ *when its target system goal is completed*:

$$trans(complete(-, B), T, i(id, A, T_i), pend, sat) \leftarrow$$
$$cur\_state(i(id, A, T_i), pend, T) \wedge T \geq T_i.$$

Axiom 7 (Joint Goals Violation). *A satisfied joint goals SN instance becomes* $violated$ *when its target system goal is completed. Being in* $satisfied$ *state implicitly indicates that the source system goal is not completed yet*:

$$trans(complete(-, B), T, i(id, A, T_i), sat, viol) \leftarrow$$
$$cur\_state(i(id, A, T_i), sat, T) \wedge T \geq T_i.$$

Axiom 8 (Semantics of Pending-based Violation). *When the case reaches an end, all pending instances are declared as violated. This is a generic axiom that applies for all pending SNs, as it attests that the case has reached it end and as a result no further events is expected to occur to move the instance from the pending to the satisfied state*:

$$trans(ev(-, d, case\_complete), T, i(-, -, -), pend, viol).$$

SN II: Metric Time Constrained Sequential Roles System Norm:

Let us now consider a time-constrained system norm. In this type of directional system norm between two system components (e.g. institutional roles A and B) where A is the source institutional role (IR) the system participant has to play the target role after it plays the source role with minimum delay (n) and maximum delay (m) time units. For example:

$$role\_role\_relation(IR\_TeamLead, IR\_Manager, (tc\_sequential\_roles, (1500, -))).$$

states that a Metric Time Constrained Sequential Roles relation holds between $IR\_TeamLead$ and $IR\_Manager$ institutional roles, expressing the requirement that a system participant can be a manager only after being a team lead for a time period of 1500 time units at least. To express this kind of system norm in EC we define the following axioms.

Axiom 9 (TC Sequential Institutional Roles Creation). *Each sequential institutional roles system norm is associated with a unique instance, created and put in the pending state waiting for*

the completion of playing role A when the case is started:

$$init\_state(i(id, start, 0), pend).$$

Axiom 10 (TC Sequential Institutional Roles Pending). *An initially pending TC Sequential Institutional Roles SN instance remains in the state pending when its source IR is played (completed). This self transition is done only to set the start time. This reference point is combined with the norm's delay n and deadline m to determine the time window, inside which the target role is expected to be played and against which we can determine the violation or satisfaction of the norm condition*:

$$trans(complete(-, A), T, i(id, A, T_i), pend, pend) \leftarrow$$
$$cur\_state(i(id, A, T_i), pend, T) \wedge T \geq T_i.$$

Axiom 11 (TC Sequential Institutional Roles Fulfilment). *A pending TC Sequential Institutional Roles SN instance is satisfied when its target institutional role is played within the time frame specified*:

$$trans(complete(-, B), T, i(id, A, T_i), pen, sat) \leftarrow$$
$$cur\_state(i(id, A, T_i), pend, T) \wedge T \geq T_i + n \wedge T \geq 0.$$

The absence of a deadline means it is effectively ∞. We check also that $T > 0$, as $T = 0$ indicates that pending is the initial state, set during the creation of this instance.

Axiom 12 (TC Sequential Institutional Roles Violation). *A pending TC Sequential Institutional Roles SN instance is violated when: (i) the target role is played before the source role, or (ii) the target role is played outside the time frame specified by the minimum and maximum delay values (n, m). In our example the maximum time delay is not specified so we limit this axiom to the first case*:

$$trans(complete(-, B), T, i(id, A, T_i), pend, viol) \leftarrow$$
$$cur\_state(i(id, A, T_i), pend, T) \wedge T < T_i + n.$$

Axiom 13 (Violation due to Deadline Expiration). *In the case of time constrained SN, the SN is violated if it is still pending after the maximum delay time passes without the completion of the target specified activity such as playing a role or sending a message ... etc. This axiom handles this case and is valid only when the maximum time delay is specified*:

$$trans(-, T, i(id, A, T_i), pend, viol) \leftarrow$$
$$cur\_state(i(id, A, T_i), pend, T) \wedge T > T_i + m.$$

6.7 Verification at Run Time

Static verification of the models is good way to check if the model complies with the requirements or not but a-priori compliance does not guarantee that the execution of the system will comply at run time. This might be because the autonomous interacting system participants deviate from the prescriptions of the model or that other external components that are designed and developed by other teams or business partners do not behave as expected, which can lead

to deviation of the whole system from its goals. To tackle this problem, online monitoring is the solution, where the behaviour of the interacting entities is checked as it happens.

Reasoning for verification is normally one of three types:

1. ***Open Reasoning:*** This type deals with partial execution traces and is able to accept more events as they happen, that why it needs to remain "open". This type is more suitable for run-time verification.
2. ***Closed Reasoning:*** In this type, a full set of events is to be verified, when an instance reaches an end of execution or a full trace of events is received. Although there are some cases that this kind of reasoning may fit for online verification, if used it leads to late/delayed discovery of system failures and system participants violations. Hence this kind of reasoning is normally used on static sets and normally associated with system model verification, during the design time.
3. ***Semi-Open Reasoning:*** This is a combination the previous two types and can be used only when the system events happen in an ascending order, so the reasoning can be closed when looks at past events and remain open when with regard to the future events.

Run-time verification as defined by Montali [2010] is the techniques that are used to verify an observed target system with respect to given requirements. Verification serves for ensuring that the observed behaviour matches specified properties as well as to recognise any instance of undesired behaviour. A basic requirement for any online verifier is that it must be able to deal with incomplete information, where the traces of events might partially fulfill some requirements at a point in time, while the rest of the requirements might be fulfilled in the future, once the full trace is complete. With regard to this partial fulfillment it is important that the verifier does not infer that this is a wrong execution, instead it is just a partial fulfilment that means the requirements have not yet been satisfied fully.

We adopt the solution of representing violations as one state of a possible set of states that the constraint can move between over time. We use as well the indirect effect modelling method to create a connection between the system norm and the events that affect it, hence the system norm constraint state changes. This approach supports continuous reasoning even after a violation happens, which is a desirable feature and a requirement for monitoring the system while it evolves in an autonomous manner. The following working example illustrates how we monitor and reason about the execution traces and MSMAS norms at run-time.

6.7.1 Monitoring Example

Let us consider the composite business process model shown in Figure 6-9. There are five Basic Business Processes, each corresponding to a Basic System Goal. This business process is a segment of an ordering system that supports the selling of goods online and offers the customer the option to pay either online or by sending a cheque. The model is designed to capture the following business requirements:

R1: After the customer communicates with the seller to place an order, the seller has to register

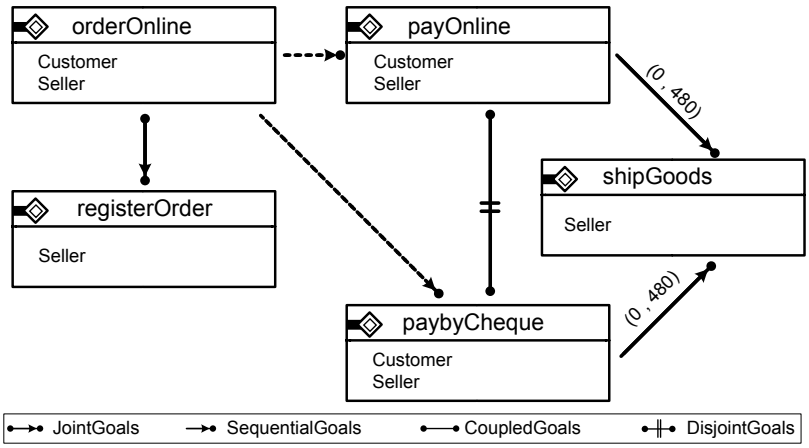


Figure 6-9: *Example MSMAS Composite Business Process Model with Goal/Goal relations*

the customer's order.

R2: The customer can pay the total selling price of his/her order either by credit card online, or by posting a cheque.

R3: The customer pays online *only* after he/she has ordered.

R4: If the customer pays online then he/she cannot pay by cheque and vice versa to prevent duplicate payment.

R5: After completion of payment, the seller must ship the goods to the customer within at most 480 time units.

These business requirements are captured and modelled by means of goal/goal system norms, expressed visually as six relations between the business process pairs as follows:

SN1: **JointGoals** relation between *orderOnline* and *registerOrder* to capture R1.

SN2: **SequentialGoals1** relation between *orderOnline* and *payOnline* to capture R2 and R3.

SN3: **SequentialGoals2** relation between *orderOnline* and *paybyCheque* to capture R2 and R3.

SN4: **DisjointGoals** relation between *payOnline* and *paybyCheque* to capture R4.

SN5: **TimeConstrainedSequentialGoals1** relation between *payOnline* and *shipGoods* to capture R5.

SN6: **TimeConstrainedSequentialGoals2** relation between *paybyCheque* and *shipGoods* to capture R5.

This set of system norms can then be expressed as:

$$goal\_goal\_relation(orderOnline, registerOrder, JointGoals).$$
$$goal\_goal\_relation(orderOnline, payOnline, SequentialGoals).$$
$$goal\_goal\_relation(orderOnline, paybyCheque, SequentialGoals).$$
$$goal\_goal\_relation(payOnline, paybyCheque, DisjointGoals).$$
$$goal\_goal\_relation(payOnline, shipGoods, (TCSequentialGoals, (-, 480))).$$
$$goal\_goal\_relation(paybyCheque, shipGoods, (TCSequentialGoals, (-, 480))).$$

Now let us consider the following execution traces where the first trace satisfies the requirements:

Trace 1:
$satisfy\_goal(orderOnline, sellerAgent), 20$
$satisfy\_goal(orderOnline, customerAgent), 21$
$satisfy\_goal(registerOrder, sellerAgent), 30$
$satisfy\_goal(payOnline, sellerAgent), 31$
$satisfy\_goal(payOnline, customerAgent), 32$
$satisfy\_goal(shipGoods, sellerAgent), 180$

Here, the customer completes orderOnline goal at time 21, then payOnline at time 32 which satisfies R2 and R3. The seller completes orderOnline at time 20, then registerOrder at time 30 which satisfies R1. The seller completes payOnline at time 31 which satisfies R2 and R3. Finally, the seller completes shipGoods at time 180 which satisfies R5. Neither the seller nor the customer start and complete paybyCheque goal within this trace, which satisfies R4. Figure 6-10(a) shows[10] the list of system norms and their state evolution according to execution trace 1. But the second trace violates the requirements:

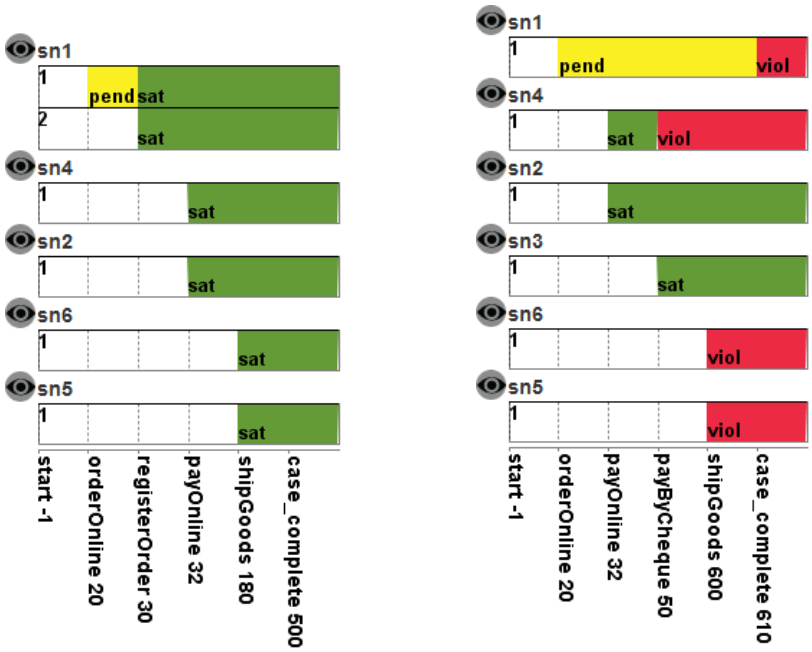
Trace 2:
$satisfy\_goal(orderOnline, sellerAgent), 20$
$satisfy\_goal(orderOnline, customerAgent), 21$
$satisfy\_goal(payOnline, sellerAgent), 31$
$satisfy\_goal(payOnline, customerAgent), 32$
$satisfy\_goal(paybyCheque, customerAgent), 50$
$satisfy\_goal(paybyCheque, sellerAgent), 50$
$satisfy\_goal(shipGoods, sellerAgent), 600$

Here, the customer completes orderOnline goal at time 21 and payOnline at time 32, which satisfies R2 and R3. Then the customer completes paybyCheque goal at time 50 which violates SN4/R4. The seller completes orderOnline goal at time 20, then payOnline at time 31 which satisfies R2 and R3. The seller completes paybyCheque at time 50 which violates SN4/R4. The seller completes shipGoods at time 600, which violates R5 because it occurs after the maximum time delay allowed by the time constrained sequential SN. Finally, the seller never completes registerOrder within this trace, so it is considered violated. Figure 6-10(b) shows the list of system norms and their state evolution according to execution trace 2.

6.8 Discussion

Since the publication of AgentLink Roadmap [Luck et al., 2005] that considered norms as key for the development of MAS, great attention was directed towards modelling MAS as normative systems. The use of norms to model MAS impacted how we think of an agent. A norm-aware agent must consider how the constraints imposed by the norms upon it and others affect not only its behaviour but also the behaviour of other agents. Agents who observe how often other agents comply with the system norms can simplify their planning, in the case that

[10]Generated using MOBUCON tool (`https://www.inf.unibz.it/~montali/tools.html`, retrieved 31 Jan 2014), using a reduced version of events (limited completion events e.g. satisfyGoal only)



(a) System Norms Evolution According to Execution Trace 1

(b) System Norms Evolution According to Execution Trace 2

Figure 6-10: *Comparison of System Norms Evolution According to Trace 1 and Trace 2*

other agents' behaviour meets the expectations, or can lead to reconsideration of the agent's own intentions and plans [Broersen et al., 2013].

The use of logic to represent the system norms is an efficient way to formally express and reason about them. Agents need to store the basic facts about the external world to be able to derive the rest by reasoning [Wooldridge, 2008]. A logic-based agent needs then a knowledge base and a set of deduction rules known as logic program. Besides taking advantage of representing system norms by means of logic, MSMAS uses system norms to provide social semantics and control over interactions within groups of system participants, where each group is an instance of an institution and where each system norm represents socially contextualised constraint on the future behaviour of one or more system participants. These future states are known as expectations, where the fulfilment or violation of them have significant impact on the system health in terms of achieving its goals. Expectations are not linked only to norms but also to contracts, commitments, agent interaction, and joint plans. Hence Broersen et al. [2013] argue that the study of techniques to formally model and reason about expectations can lead to a unified treatment of commitments and norms.

A wealth of research has taken place to formalise normative systems and institutions structure, such as [Andrighetto et al., 2012] and the work of García-Camino et al. [2005], and Boella and der Torre [2006]. While Alberti et al. [2006] have proposed modelling agent interaction protocols using an explicit representation of expectations, in their work they expressed communication protocols using logical rules defining expectations of agents' communicative acts

based on current and past communicative acts. They also used $\mathcal{S}$CIFF [Alberti et al., 2008] as an abductive proof procedure to verify agents' compliance with protocols.

In our view, an essential part of reasoning about expectations is time, and while most formalisms do not have temporal operators, researchers have considered two concepts in relation to time: the 'validity time' where a state has a truth value in terms of normative context, such as a current obligation and 'reference time' where an obligation, prohibition or permission comes into effect at some point in the future. MSMAS's chosen formalisation of system norms relies on expectations and recognises the importance of time, hence we adopt ConDec^{++} for the visual design and express the system norms semantics using $\mathcal{S}$CIFF. Furthermore, MSMAS use of norms with an underlying metamodel following model-driven development approach, is part of the active research of Metamodels for normative MAS. The Alive project Aldewereld et al. [2010] used model-driven development to deploy agents and services in the context of an organisational model defined in Opera, which expresses norms in terms of states to be maintained or avoided. However, Opera lacks an activity model, making run-time validation difficult. The very detailed formalisation of EIDE [d'Inverno et al., 2012] takes what is arguably a more low-level approach to capturing the quite complex semantics, particularly those associated with scenes and transitions, using the Z specification language. While this provides a high degree of precision, there is no associated metamodel, so although there are tools to generate code automatically, they do not have that formal backing. Ghorbani Ghorbani et al. [2013] provide a different perspective, in which a meta-model is defined and used for model-driven development in the context of agent-based simulations, informed by Ostrom's IAD framework. While this emphasises the role of norm and institution in the governance of agent behaviour, the metamodel reflects only the research objectives of modelling social structures and their evolution.

Verifying requirements at run time is widely recognised as important aspect of modern systems, given the increased complexity of MAS and increased dynamic nature of business requirements. Some approaches focus on traceability between requirements and source code and/or between design and source code [Grechanik et al., 2007], while others focus on traceability between requirements and architecture [Goknil et al., 2014] allowing for generation and validation of traces by using requirements relations. In contrast, MSMAS utilises logic-based languages to encode formally the system norms that represent the system requirements into the designed models and become an integral part of the deployment. Hence traceability of these requirements during design time to verify the correctness of the designed models as well as monitoring the system execution traces and verifying that the execution trace meets the requirements are achievable tasks. The verification phase of MSMAS has some similarities with the work of Balke et al. [2012]. They propose for run-time verification a normative component with the sole function of monitoring agents' actions and verifying their legitimacy either they were allowed or not from a normative perspective. The normative component, in Balke et al. [2012]'s work, uses a run-time model extracted from the design time model and

is limited to only the normative information. While MSMAS's formal model being limited to only the normative information, the same model is suitable for both design time and run-time verification.

Monitoring and runtime compliance checking using an abductive logic programming-based proof procedure was proposed in Chesani et al. [2010], while the preliminary investigation of the EC-based axiomatisation of ConDec was first proposed in Chesani et al. [2009b]. Chesani et al. [2009a] used an extension of the event calculus based on $\mathcal{S}$CIFF for the specification of the social semantics of agent interactions based on the theory of commitments. Their work was aimed at supporting run-time verification using the notion of expectations. Monitoring agent compliance with institutional rules/system norms was also studied by Cardoso and Oliveira [2007] and more recently by Alvarez-Napagao et al. [2011], who presents a formalisation of the life cycle of regulative and substantive norms and a supporting reasoner.

MSMAS models are derived from the identified requirements, where system goals are linked with business activities that are associated with institutional roles and one or more system norms are added to capture one or more requirements. The encoding of these requirements as system norms allows MSMAS to use these normative specifications to verify the maintenance of system requirements during execution in a similar approach to the work of Chesani et al. [2009a], although the formalisation is different. We have adopted this approach due to the availability of scalable support tools ready for integration.

Against this background we suggest that: (i) the novel combination of a business-oriented institutional meta-model, designed to support self-management in use, (ii) with a design- and run-time formal proof mechanism, (iii) provides a new perspective on formal software engineering of MAS and contributes towards the evolution of and the debate on the application of meta-models and model-driven development in MAS.

6.9 Chapter Summary

Building a system that handles business critical functions requires implementing a degree of self-management to allow the system to detect any deviation from its goals, and apply corrective actions, to maintain a high level of adherence to requirements and resilience. Verification of designed system models helps to ensure that the models are a fair representation of the design goals, while verifying at run time validates that the running system traces are inline with the requirements. Monitoring the execution trace is also one of the means for the system to be able to self-manage. MSMAS uses logic to represent formally the system requirements that are captured as system norms. The system norms are expressed in CLIMB to enable the use of $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF proof procedures for static verification of MSMAS models, while the same norms are expressed as EC theories for run-time verification.

CLIMB uses a number of events as terms with both explicit, implicit and quantitative notion of time. The system norms are expressed as integrity constraints that captures the dynamic

nature of the system by specifying, at its simplest form, what is expected to happen when an event happens. The static knowledge of the system is represented and stored as a knowledge base. We have defined a number of event types associated with MSMAS system norms and we mapped our defined system norms in CLIMB. Reasoning about MSMAS models is then possible by the examination of the fulfillment of synthesised trace to satisfy a property. We have explained how to verify MSMAS models at design time using $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF proof procedures.

EC uses a number of triggering events that causes fluents to move between their life cycle states. The system is expressed as a set of domain-independent theories to specify the activity life cycles and a set of domain-specific theories that capture the system norms transition life cycles. Deductive Reasoning is then possible by taking an execution trace and the EC domain theory to infer the validity intervals of fluents. We define two simplified lifecycles for MSMAS system norms represented as non-atomic activities. We also express MSMAS norm transitions as EC domain theory and illustrate them by means of a monitoring example of two possible execution traces to show how to reason about MSMAS norms and discover any violation of norms.

CHAPTER 7

EVALUATION

Multi Agent Software Engineering is based on concepts and principles for constructing complex distributed intelligent systems in which agents are the main building block of the system. The main goal of multiagent development methodologies is to define a process for designing and specifying and implementing such systems. Although, few multiagent development methodologies are mature enough and complete, compared to traditional software development methodologies such as Object-Oriented, we have found many useful evaluation frameworks for evaluating MAS methodologies and it has become clear that there is no obvious need to develop our own framework. Add to that, the additional credibility of the assessment arising from the use of an existing and recognised approach. Thus, we briefly consider several existing frameworks and choose one that we believe can provide an objective assessment for our methodology, and can identify its weaknesses and its strengths when compared to other existing methodologies.

We understand as well that there is no complete evaluation framework, so our chosen framework might not be perfect, but our objective is to provide a fair comparison to highlight where MSMAS differs from other methodologies. The true assessment of its success remains to be proven over time through the popularity it might gain or through the interest it might receive from its potential users.

In this chapter we introduce our chosen evaluation framework, apply its assessment criteria to MSMAS, and present the results alongside other results of evaluating some other methodologies. We end the chapter with a discussion of the results. The chapter is organised as follows: Section 7.1 gives the rationale, by reference to several alternative frameworks, for the chosen framework, Section 7.2 describes the evaluation framework in some detail, Section 7.3 presents MSMAS evaluation results according to the evaluation framework; and finally, Section 7.4 concludes this chapter.

7.1 Introduction

There are numerous MAS methodologies, that have been proposed over the last fifteen years, which creates the problem of choice for MAS developers in deciding which method works better for them or for their application. A structured approach to make that choice might be the use of a feature analysis framework, but those frameworks that exist are aimed at evaluating conventional system development methodologies and are not obviously suitable for evaluating MAS development methodologies. In response to this need, a number of agent-oriented evaluation frameworks were proposed such as [Shehory and Sturm, 2001], [O'malley and DeLoach, 2002] and [Cernuzzi et al., 2002] and more recently [Abdelaziz et al., 2007] and [Beydoun et al., 2012].

However, these frameworks do not address the evaluation of MAS development methodologies from the software engineering point of view [Tran et al., 2003]. In consequence, we choose the proposed framework of Tran et al. [2003] because they present a comprehensive, multidimensional framework that allows for the evaluation of MAS development methodologies from the general view of software engineering as well as from the view that is specific to multiagent systems. We have found also that many of the recently proposed feature analysis evaluation frameworks are very similar to Tran et al. [2003]'s but less comprehensive. The framework provides a list of evaluation criteria that were established after considering various feature analysis frameworks for evaluating conventional system development methodologies such as [Wood et al., 1988], [Jayaratna, 1994] as well as various MAS evaluation frameworks such as [Shehory and Sturm, 2001], [O'malley and DeLoach, 2002] and [Cernuzzi et al., 2002]. In the next section, we summarise the framework then present the evaluation of MSMAS features, based on the framework criteria.

7.2 The Evaluation Framework

The Feature Analysis Framework for Evaluating MAS Development Methodologies (MASDM) is proposed by Tran et al. [2003], where they adopt evaluation criteria from other software development methodologies evaluation frameworks in general and from multiagent methodologies evaluation frameworks in particular. MASDM focuses on those criteria that are centered around the capabilities and usefulness of the software methodology. Tran et al. [2003] added also more criteria that were not accounted for in other evaluation frameworks. MASDM was used in evaluating a number of MAS methodologies [Tran et al., 2005], [Al-Hashel et al., 2007], also some other frameworks used very similar criteria that we believe were inspired by it such as Gholami et al. [2010] and Kumar and Goyal [2012].

Figure 7-1 shows the main components of the evaluation framework, of which there are four:

- **Process Related Criteria:** that focus on evaluating the methodology's support for MAS development process. There are fifteen identified criteria as shown in Table 7.1. The

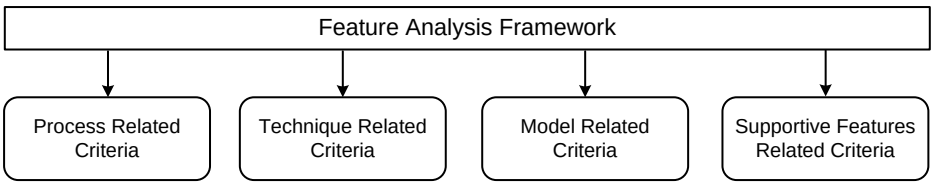


Figure 7-1: *Feature Analysis Evaluation Framework Structure [Tran et al., 2003]*

most important ones are: the coverage of the life cycle (which phases are covered?), support for verification (are there integrated steps to verify and validate the models and other artifacts?), and usability of the methodology (are the steps, and graphical notation are easy to follow?).

- **Technique Related Criteria:** to evaluate the techniques the methodology uses to develop the MAS. It contains five criteria as shown in Table 7.2. The most important ones are: ease of understanding of techniques, and availability of examples.
- **Model Related Criteria:** to assess the methodology's models capabilities. This component contains twenty two criteria, as shown in Table 7.3. The most important criteria are: expressiveness (how well the model capture the concepts?), completeness does it include all need concepts to describe the system, model derivation(is possible to transforming the models or derive model from another?), abstraction (producing models at various levels of abstraction), and communication ability (is able to model communication).
- **Supportive Feature Criteria:** are eight criteria to evaluate high-level methodological capabilities, as shown in Table 7.4. Among them the most important in our view are: open system and scalability, dynamic structure (does it support open structure and reconfiguration of the system?), support for self-interested agents, and support for conventional objects and ontology.

Each criterion in the framework is accompanied by one or more evaluation questions to be answered as shown in Table 7.1, 7.2, 7.3, and 7.4. In addition, the *Steps in the development process* under the Process Related Criteria and the *Concepts* in the Model Related Criteria each require a rather more comprehensive assessment. This can be done through the use of a list of standard process steps and concepts that are shown in Table 7.5.

Feature Analysis Framework:	**Process Related Criteria**
Development life cycle	Which development life cycle style best describes the methodology? Is it waterfall, iterative, or spiral?
Coverage of the life cycle	What phases of the development life cycle are covered by the methodology (e.g. analysis, design, and implementation?
Development perspective	What development perspective is supported? Is it top-down, bottom-up, or hybrid?
	Continued on next page

Table 7.1 – continued from previous page

Feature Analysis Framework:	Process Related Criteria
Application domain	Is the methodology applicable to a specific or multiple application domains?
Size of MAS	What size of MAS is the methodology suitable for?
Agent nature	Does the methodology support agents of any type (i.e. heterogeneous, homogeneous, or mobile agents), or only a particular type or agents?
Support for verification	Do the methodology phases or sub-phases contain a validation or a verification process? Can the design be verified or validated against some rules to check its correctness.
Steps in the development process	What development steps and tasks are supported by the methodology?
Notational components	What models and diagrams are generated from each process step?
Comments on the overall strengths/weaknesses of each step	A criterion to be used by the evaluator to record any comments on a process step that does not fit under other criteria.
Ease of understanding of the process steps	Are the process steps easy to understand?
Usability of the methodology	Are the process steps easy to follow? Are the graphical diagrams, and key notations clear enough to allow the developer to progress smoothly in development processes?
Definition of inputs and outputs	Are inputs and outputs to each process step clearly defined, and explained with examples?
Refinability	Does the methodology allow for refining the methodology's models through clear steps and gradual stages to reach an implementation, or at least has steps to connect the implementation level with the design specification?

Continued on next page

Table 7.1 – continued from previous page

Feature Analysis Framework:	Process Related Criteria
Approach towards MAS development	Is the methodology: • A generic MAS development approach (e.g. Object-Oriented-based or knowledge-engineering based)? • An approach towards using (role) in MAS development (e.g. does it deploy the concept of (role) in the MAS analysis)? • An approach in role identification, if the methodology uses (goal) in MAS development? Is it goal-oriented, behaviour-oriented, or organisation-oriented?

Table 7.1: *Feature Analysis Framework for Evaluating MASDM: Process Related Criteria [Tran et al., 2003]*

Feature Analysis Framework:	Technique Related Criteria
Availability of techniques and heuristics	• What are the techniques to perform each process step? • What are the techniques to produce each notational component (i.e. modeling techniques)?
Comments on the strengths / weaknesses of the techniques	An additional criterion to allow the evaluator to record any comments on the techniques to perform each step or to produce each model.
Ease of understanding of techniques	Are the techniques easy to understand?
Usability of techniques	Are the techniques easy to follow?
Provision of examples and heuristics	Are examples and heuristics of the techniques provided?

Table 7.2: *Feature Analysis Framework for Evaluating MASDM: Process Related Criteria [Tran et al., 2003]*

Feature Analysis Framework:	Model Related Criteria
Concepts	What concepts are the methodology's models capable of expressing?

Continued on next page

Table 7.3 – continued from previous page

Feature Analysis Framework:	Model Related Criteria
Expressiveness	How well can each model express these concepts? (e.g. is each model capable of capturing the concept at a great level of detail, or from different angles?)
Completeness	Are all necessary agent-oriented concepts that describe the target MAS captured by the methodology's models?
Formalisation / Preciseness of models	Are notation (syntax) and semantics of models clearly defined?
Model derivation	Does there exist explicit process/logic and guidelines for transforming models into other models, or partially creating a model from information present in another?
Consistency	• Are there rules and guidelines to ensure consistency between levels of abstractions within each model (i.e. internal consistency), and between different models? • Are presentations expressed in a manner that allows for consistency checking between them?
Complexity	is there a manageable number of concepts expressed in each model/diagram?
Ease of understanding of models	Are the models easy to understand?
Modularity	Does the methodology and its models provide support for modularity of agents?
Abstraction	Does the methodology allow for producing models at various levels of detail and abstraction?
Autonomy	Can the models support and present the autonomous feature of agents (i.e. the ability to act without direct intervention of humans or others, and to control their own states and behaviours)?
Adaptability	Can the models support and present the adaptability feature of agents (i.e. the ability to learn and improve with experience)?
Cooperative behaviour	Can the models support and present the cooperative behaviour of agents (i.e. the ability to work together with other agents to achieve a common goal)?

Continued on next page

Table 7.3 – continued from previous page

Feature Analysis Framework:	Model Related Criteria
Inferential capability	Can the models support and present the inferential capability feature of agents (i.e. the ability to act on abstract task specifications)?
Communication ability	Can the models support and present "knowledge-level" communication ability (i.e. the ability to communicate with other agents using language resembling human-like speech acts)?
Personality	Can the models support and present the personality of agents (i.e. the ability to manifest attributes of a "believable" human character)?
Reactivity	Can the models support and present reactivity of agents (i.e. the ability to selectively sense and act)?
Temporal continuity	Can the models support and present temporal continuity of agents (i.e. persistence of identity and state over long periods of time)?
Deliberative behaviour	Can the models support and present deliberative behaviour of agents (i.e. the ability to decide in a deliberation, or proactiveness)?
Concurrency	Does the methodology allow for producing models to capture concurrency (e.g. presentation of concurrent processes and synchronisation of concurrent processes)?
Human Computer Interaction	Do the models present human users and the user interface?
Models Reuse	Does the methodology provide, or make it possible to use, a library of reusable models?

Table 7.3: *Feature Analysis Framework for Evaluating MASDM: Model Related Criteria [Tran et al., 2003]*

Feature Analysis Framework:	Supportive Feature Criteria
Software and methodological support	Is the methodology supported by tools and libraries (e.g. libraries of agents, agent components, organizations, architectures and technical support)?
Open systems and scalability	Does the methodology provide support for open systems and scalability (e.g. the methodology allows for dynamic integration/removal of new agents/resources)?

Continued on next page

Table 7.4 – continued from previous page

Feature Analysis Framework:	Supportive Feature Criteria
Dynamic structure	Does the methodology provide support for dynamic structure? (i.e. the methodology allows for dynamic reconfiguration of the system)?
Agility and robustness	Does the methodology provide support for agility and robustness (e.g. the methodology captures normal processing and exception processing, provides techniques to analyze system performance for all configurations, or provides techniques to detect/recover from failures)?
Support for conventional objects	Does the methodology cater for the use/integration of ordinary objects in MAS (e.g. the methodology models the agents' interfaces with objects)?
Support for mobile agents	Does the methodology cater for the use/integration of mobile agents in MAS (e.g. the methodology models which/when/how agent should be mobile)?
Support for self-interested agents	Does the methodology provide support for MAS with self-interest agents (whose goals may be independent or enter in conflict with other agents' goals)?
Support for ontology	Does the methodology cater for the use/integration of ontology in MAS (i.e. ontology-driven agent systems)?

Table 7.4: *Feature Analysis Framework for Evaluating MASDM: Supportive Feature Criteria [Tran et al., 2003]*

7.3 MSMAS Evaluation

Evaluating MSMAS features in isolation from other available methodologies will not be sufficient to understand how strong or weak the MSMAS is. For that reason, we evaluate MSMAS alongside some other methodologies from the ones we highlighted in Section 2.3, namely GAIA (GA) [Wooldridge et al., 2000a], TROPOS (TR)[Bresciani et al., 2004], PROMETHEUS (PR) [Padgham and Winikoff, 2002], and SONIA (SO) [Alonso et al., 2005]. The first three are well known and there is wealth of literature that shows they are widely used and referred to in academic research, while SONIA represents a more recent attempt that makes use of ontology. The criteria assessments are summarised in tables and the narrative criteria are discussed in the following sections. Then an overall discussion is presented in Section 7.4 in the end of this chapter.

Steps	Identify system goals Identify system roles Develop system use cases/scenarios Identify system functionality Identify design requirements Identify agent classes Specify agent interaction pathways Define exchanged messages Specify interaction protocols Specify contracts/commitments Specify ACL	Specify conflict resolution mechanisms Define agent architecture Define agents' mental attitudes (goals, plans, beliefs...) Define agents' interface (capabilities, services...) Fulfil agent architecture Define system architecture Specify organizational structure Specify group behaviour	Specify agent relationships (inheritance, aggregation & association) Specify co-existing entities Specify environment facilities Specify agent-environment interaction Instantiate agent classes Specify agent instances location
Concepts	System goals System roles System functionality Task responsibilities/procedures Design requirements Use case/scenarios Agent classes Agent instances Agent's knowledge/beliefs Agent's plans Agent's goals Agent's roles	Agent's functionality Precepts/Events Agent mobility Interaction pathways Exchanged messages Interaction protocols Interaction constraints Conflict resolution mechanisms Contracts/commitments ACL Ontology Agent inheritance	Agent aggregation Agent association Co-existing entities Environment facilities Organizational structure Group behaviour Agent-environment interaction Environment characteristics Agent architecture System architecture Location of agent instances Sources of agent instances

Table 7.5: *Feature Analysis Evaluation Framework List of Standard Steps and Concepts [Tran et al., 2003]*

7.3.1 Process Related Criteria

This section considers the aspects related to the development process of MASs. We look at the applicability of the methodology steps, its stages and its development approach. We have considered earlier in Section 2.3 the development steps of each methodology, and listed a summary of their models. Hence we limit the presentation of the comparative analysis results in Table 7.6 only to the remaining criteria.

Evaluation Criteria	**GA**	**PR**	**TR**	**SO**	**MSMAS**
Development life cycle	Iterative across all phases	Iterative across all phases	Iterative and incremental	Iterative	Iterative and incremental
Coverage	A, D	A, D	A, D	A, D	A, D, I
Development approach	Top-down	Bottom-up	Bottom-up	Hybrid	Hybrid
Application domain	Any	Any	Any	Any	Any
Size of MAS	$\leq$ 100 agents	Not specified	Not specified	Not specified	Not specified

Continued on next page

Table 7.6 – continued from previous page

Evaluation Criteria	GA	PR	TR	SO	MSMAS
Agent nature	H[1]	BDI-Like	BDI-like	BDI-Like	BDI-like
Support for verification	No	Yes	No	No	Yes
Ease of understanding of process steps	High	High	Medium	High	High
Usability of the methodology	Medium	High	Medium	Medium	High
Refinability	Yes	Yes	Yes	Yes	Yes
Approach towards MAS development	OO - RO	OO - NRO	GO - NRO	OO - OrO	GO - RO - OrO

Table 7.6: *Comparative Analysis Results: Process Related Criteria*

Our assessment indicates that all the MAS methodologies under study adopt an iterative software development life cycle. MSMAS allows for iterative refinements for its models specifically e.g. the refinement of system goals and defining the various system norms. All methodologies but MSMAS, cover only the Analysis (A) and Design (D) phases, while MSMAS covers the Analysis, Design and Implementation (I) phases. Considering the development perspective, GAIA is top-down, PROMETHEUS, and TROPOS are bottom-up, while SONIA and MSMAS are hybrid.

An AOSE methodology is a top-down if it starts from the analysis of high level elements such as system goals, problem statement, and organisational structure then proceeds to the identification the designing of agents and other system components. In contrast, a bottom-up AOSE methodology begins with the analysis of low level behaviours and system activities then it packages and groups them to compose agents. MSMAS' approach is a hybrid, which means it integrates both approaches by identifying agents and other system participants from the consideration of both high level system goals/organization, and low level system tasks and responsibilities.

It is not surprising that most MAS methodologies are suitable for most application domains. Only GAIA is suitable for heterogeneous agents while the remaining methodologies support BDI-like agents. With regard to their support to verification, only Prometheus and MSMAS are supportive of verification and validation processes, since they provide rules and guidelines to assist the system developers in verifying and validating the developed models. MSMAS is the only methodology that supports the verification of requirements at runtime given its

[1]Heterogeneous

use of system norms and their formal representation. With regard to usability and ease of understanding, we believe these criteria have to be examined through a user evaluation process, because our perception, especially of the ease of understanding, might be biased. We perceived all methodologies except TROPOS to be easy to understand, although TROPOS is detailed in description, we find the length of its steps makes it harder to follow. We find PROMETHEUS and MSMAS are better in terms of usability, where their notations and diagrams are clear. They both also allow for smooth progress during the design process and have a supporting tool.

With regard to refinability, all methodologies have a clear route to refine their models. Considering the approaches to MAS development, our assessment reveals that every methodology supports more than one approach. GAIA supports both Object Oriented (OO) and Role Oriented (RO), PROMETHEUS supports OO and Non Role Oriented (NRO) as it relies on other constructs such as scenarios and interactions to develop agents. TROPOS supports Goal Oriented (GO) and NRO, SONIA supports OO and Organisation Oriented (OrO), and MSMAS supports GO, RO, and OrO.

7.3.2 Technique Related Criteria

Technique Related Criteria focus on (i) the availability of clearly defined techniques to produce each model and notational components and if this is available across all steps (ii) technique usability, which assesses the availability of supporting tools and templates, and (iii) ease of understanding to check for unambiguous syntax and semantics and how easy it is to learn to use the methodology techniques. We show in Table 7.7 the comparative analysis results, where (H) stands for High, (M) for Medium, and (L) for Low. We show in the last column ($\pm$) a rating of MSMAS on each criterion based on the following rating system:

+ means MSMAS slightly better than other methodologies under study e.g. only one other methodology as good as MSMAS, or MSMAS can support this but not explained how.

++ means MSMAS is better than all other methodologies under study.

- means MSMAS is worse than other methodologies under study.

- - means this is a weakness or missing feature in all methodologies under study including MSMAS.

= means MSMAS is as good as other methodologies under study.

Evaluation Criteria	GA	PR	TR	SO	MSMAS	$\pm$
Availability of Techniques and Heuristics	H	H	H	H	H	=
Ease of Understanding	H	H	H	H	H	=
Usability of Techniques	H	H	H	M	H	=
Provision of Examples and Heuristics	Yes	Yes	Yes	Yes	Yes	=

Continued on next page

Table 7.7 – continued from previous page

Evaluation Criteria	GA	PR	TR	SO	MSMAS	±

Table 7.7: *Comparative Analysis Results: Technique Related Criteria*

With regard to the technique related criteria, all the methods under study score highly, except SONIA which does not have a supporting tool.

Comments on the Strengths/Weaknesses: In this part of the evaluation we used our own experience to evaluate PROMETHEUS, SONIA and MSMAS, combined with the results from the work of Kumar and Goyal [2012]. We observe that all five methodologies provide good support at each step either implicitly or explicitly. They also provide various constructs to model their concepts. GAIA provides a role model for identifying system tasks implicitly and an agent model for specifying agent classes. Other techniques provided by GAIA include a set of models that are used at different steps. PROMETHEUS provides a large number of diagrams such as goal diagrams, interaction diagrams and protocols and capability diagrams, as well as a number of descriptors such as agent class descriptor. TROPOS offers a smaller set of diagrams, such as actor diagrams, plan diagrams and sequence diagrams. With regard to usability, only PROMETHEUS provides a tool to draw diagrams and check model and design consistency. Other methodologies do not offer such support, although Kumar and Goyal [2012] report that developers claim that the notations and symbols are fairly easy to understand. MSMAS has a supporting tool to create the visual models and their descriptors. It also allows for exporting the system models in multiple formats, that can be used for verification and monitoring of deployed system.

7.3.3 Model Related Criteria

Designed models are visual representations that capture and highlight the most important aspects of the problem under modelling. A good model design allows for the translation or mapping from that model to another form without loss of details and allows for consistent understanding between different groups of people. Evaluating MAS development methodology includes consideration of its group of models, how these models can be created and how expressive they are of the aspects they visualise. The relationships between models are another factor that strengthens the methodology or weakens it. If there is relation between different system components that belong to different models, then a relationship between these models should exist, otherwise there is a risk of losing some semantics. The number of methodology models can also affect its complexity: the larger that number gets the more complex it becomes in relation to its applicability, learning, and proficiency. This section considers the model-related aspects of MAS methodologies under comparison, Table 7.8 shows the comparative analysis results.

Evaluation Criteria	GA	PR	TR	SO	MSMAS	±
Completeness	2	3	3	2	4	+ +
Formalisation / Preciseness	High	High	High	Medium	High	=
Model Derivation	Yes	Yes	Yes	Yes	Yes	=
Consistency	Yes	Yes	Yes	Yes	Yes	=
Complexity	M	H	M	H	M	=
Ease of Understanding	5	3	4	4	5	=
Modularity	Yes	Yes	Yes	Yes	Yes	=
Abstraction	Yes	Yes	Yes	Yes	Yes	=
Autonomy	Yes	Yes	Yes	Yes	Yes	=
Adaptability	No	No	No	No	No	- -
Cooperative Behaviour	Yes	Yes	Yes	Yes	Yes	=
Inferential Capability	No	Yes	Yes	Yes	Yes	=
Communication Ability	No	Yes	Yes	Yes	Yes	=
Personality	No	No	No	No	Yes	+ +
Reactivity	Yes	Yes	Yes	Yes	Yes	=
Temporal Continuity	No	No	No	No	Yes	+ +
Deliberative Behaviour	Yes	Yes	Yes	Yes	Yes	=
Concurrency	No	No	No	No	Limited	+
Human Computer Interaction	No	Yes	No	No	Yes	+
Models Reuse	Yes	Yes	Yes	Yes	Yes	=

Table 7.8: *Comparative Analysis Results: Model Related Criteria*

In comparison to other methodologies, MSMAS captures and represents the highest number of steps and concepts, as shown in Table 7.11 and Table 7.10, which indicates that MSMAS is highest with regard to the completeness criterion. Considering Formalization/Preciseness criterion, all methodologies under comparison offer detailed explanations of model notation and semantics.

All methodologies have clear steps and techniques to support the transforming of their models to into other models. PROMETHEUS and MSMAS do better with regard to model derivation criterion as some parts of their models are contributing to the creation of other models. All methodologies models can be assessed with regard to consistency, PROMETHEUS and MSMAS offer tools that allow for checking the models consistency automatically. We consider PROMETHEUS and SONIA are more complex to use, while the other three methodologies are less complex. This is also reflected in ease of understanding where GAIA and MSMAS are the easiest to understand their models and notations.

All five methodologies offer various levels of details and abstractions and support agent modularity. They all support as well autonomy, agent cooperative behaviour, reactivity, and

model reuse. Apart from GAIA, they support the inference, communication ability, and deliberative behaviour. MSMAS distinguishes itself by its support for personality (being the ability to model human like actors) and temporal continuity, where the system designer can include human actors in his models, and adding specifications that enable the persistence of agent roles/identities over long period of time. None of methodologies except MSMAS support concurrency, and only PROMETHEUS and MSMAS support the representation of humans in their models, which allow for defining interfaces from human/computer interactions. Finally all five methodologies have a good degree of expressiveness, where each model of their models captures the details of its concepts, PROMETHEUS and MSMAS use text descriptors that provide great details on each system component/concept.

7.3.4 Supportive Feature Criteria

This section considers the supportive feature aspects of MAS methodologies under comparison. Here, we observe the extent of support and availability of software tools to support the users of the methodology. The list of features in this section are considered as indicators of the development quality and performance measures of the methodologies under study. Table 7.9 shows the comparative analysis results where the tick (✓) is used to indicate that the MAS methodology supports this criterion and when left blank means it is not supported by the methodology.

Evaluation Criteria	GA	PR	TR	SO	MSMAS	±
Software and methodological support		✓			✓	+ +
Open systems and scalability	✓			✓	✓	=
Dynamic structure					✓	+ +
Agility and robustness		✓			✓	+
Support for conventional objects		✓			✓	+
Support for mobile agents						- -
Support for self-interested agents	✓		✓	✓	✓	=
Support for ontology					✓	+ +

Table 7.9: *Comparative Analysis Results: Supportive Feature Related Criteria*

With respect to the availability of software tool, only PROMETHEUS and MSMAS offer a tool that supports the designing and modelling processes. GAIA, SONIA and MSMAS give some attention to openness and scalability, while PROMETHEUS and TROPOS do not offer much support for these features. Only GAIA and MSMAS support the modelling of adaptive systems.

Only PROMETHEUS and MSMAS provide support for agility and provide techniques to measure the system performance, as well as allow for the use and integration of ordinary objects. all methodologies do not support mobility of agents in terms of expressing whether agents are mobile and how and where they migrate etc. All methodologies support self-interested agents except PROMETHEUS, while MSMAS even offers a way to specify which goals are conflicting, coupled, ordered or independent. Finally, out of the five methodologies under study, only MSMAS supports the use of ontologies, and allows for exporting the system specifications as RDF/OWL file.

7.3.5 Support for Steps in the Development Process

We use the list of standard MAS development steps shown in Table 7.5 as a checklist to compare the five MAS methodologies. The support of each methodology for each step is assessed on a 5-point scale:
0: no support is provided
1: the step is not explicitly included but it can be achieved by other means
2: the step is included but no techniques or examples are provided
3: the methodology provides techniques for performing the step
4: the step is discussed with techniques and examples
Using the rating allows us to assess the methodologies under study and identify how they compare to each other in terms of the completeness and coverage of the standard steps. Table 7.10 shows the results.

Evaluation Criteria	GA	PR	TR	SO	MSMAS	±
Identify system goals	0	0	4	0	4	+
Identify system roles	4	0	0	0	4	+
Identify system functionality/task	4	4	4	4	4	=
Develop use cases/scenarios	0	4	4	0	4	=
Produce sequence diagrams	0	0	0	0	0	- -
Identify design requirements	0	0	4	0	4	+
Identify agent classes	4	4	4	4	4	=
Specify agent interaction pathways	4	4	4	4	4	=
Define exchanged messages	0	1	0	0	4	+
Specify interaction protocols	0	4	4	1	4	=
Specify contracts/commitments	0	0	0	0	0	- -
Specify conflict resolution mechanisms	0	0	0	0	0	- -

Continued on next page

Table 7.10 – continued from previous page

Evaluation Criteria	GA	PR	TR	SO	MSMAS	±
Specify coordination/control regime	0	0	1	2	1	=
Specify agent communication language	0	0	0	0	4	+ +
Define agent architecture	0	0	4	0	0	-
Define agent mental attributes	0	4	4	1	3	-
Define agent behavioural interface	4	4	4	4	4	=
Define system architecture / organisational structure	0	0	1	4	4	+
Specify dynamic agent group formulation / dissolution	0	0	0	0	2	+
Specify agent relationships (e.g. inheritance, & association etc)	4	0	3	1	3	=
Specify co-existing non-agent entities	4	2	4	0	4	=
Specify infrastructure / environment facilities	0	0	0	0	2	+
Specify agent-environment interaction mechanism	0	4	0	0	2	-
Instantiate agent classes	1	0	1	0	2	=
Specify agent instances location	0	0	0	0	0	- -

Table 7.10: *Comparative Analysis Results: Support for Steps in the Development Process*

Rating the support of steps, shows that MSMAS is the most complete methodology, where it scored 68 out of maximum possible score of 100. While SONIA scores the lowest at only 25 points. GAIA scores 29 points, PROMETHEUS scores 35 and TROPOS is the most complete after MSMAS with a score of 50 points. We discuss in more detail the weaknesses and strengths of MSMAS based on its support of steps in Section 7.4.

7.3.6 Support for Concepts of MAS Models

We use the list of standard MAS development concepts shown in Table 7.5 as a checklist to compare the five MAS methodologies. The tick (✓) used to indicate that the MAS methodology represents or captures that concept in its models.

Concepts	GA	PR	TR	SO	MSMAS	+/-
System goals			✓		✓	+
System roles	✓				✓	++
System functionality/tasks	✓	✓	✓	✓	✓	=
Task responsibilities/Procedures	✓		✓	✓	✓	=
Design requirements			✓	✓	✓	=
Use case/Scenarios		✓			✓	+
Agent classes	✓	✓	✓	✓	✓	=
Agent instances	✓	✓	✓	✓	✓	=
Agent's roles	✓		✓		✓	=
Agent's functionality	✓	✓	✓	✓	✓	=
Agent's knowledge/Beliefs		✓	✓	✓	✓	=
Agent's plans		✓	✓		✓	=
Agent's goals			✓		✓	+
Agent's capabilities		✓	✓			-
Agent Mobility						-
Interaction pathways	✓	✓	✓	✓	✓	=
Exchanged messages		✓			✓	++
Interaction protocols		✓	✓		✓	+
Interaction constraints					✓	++
Conflict resolution mechanisms						+
Contracts/commitments						+
Ontology					✓	++
Inheritance						-
Aggregation	✓		✓	✓	✓	=
Association	✓		✓	✓	✓	=
Co-existing non-agent entities		✓	✓		✓	=
Environment facilities	✓		✓		✓	=
Organisational Structure			✓	✓	✓	=
Agent-environment interaction		✓			✓	+
Environment characteristics					✓	++
Agent architecture			✓		✓	++
System architecture			✓	✓	✓	=
Location of agent instances						-
Sources of agent instances						-

Table 7.11: *Comparative Analysis Results: Support for Concepts of MAS Models*

With regard to the support of concepts, MSMAS proved to be more expressive as it sup-

ports 29 concepts in its models out of a total of 43 concepts. GAIA supports the lowest number of concepts (11), followed by SONIA which supports 12 concepts, then PROMETHEUS that supports 13 concepts. TROPOS is the closest to MSMAS as it supports 21 concepts. We discuss in more detail the weaknesses and strengths of MSMAS based on its support of concepts in Section 7.4.

7.4 Discussion and Conclusion

We aim, by the comparative study presented in this chapter, at highlighting the differences between MSMAS and other methodologies, and assessing the strengths and weaknesses of MSMAS.

Following the main components of the evaluation framework, the results of the *Process Related Criteria*, shows that MSMAS is as good or better than other methodologies across all criteria. MSMAS is better in terms of coverage due to its inclusion of the implementation phase. More importantly, MSMAS's support for verification at both design and runtime puts it ahead of other methodologies.

With regard to the *Technique Related Criteria*, our analysis results in Table 7.7 show that all methodologies support all criteria. However, we believe MSMAS is slightly better in terms of ease of understanding due to its use of common business process modelling concepts which ease its learning curve. The results of *Model Related Criteria* analysis show MSMAS is as good or better than other methodologies on all model related criteria. MSMAS is more complete than others, which is established by the number of steps it supports as shown in Table 7.10. MSMAS is also better in terms of *Personality*, which assesses if the methodology can present attributes of a "believable" human, because the inclusion of human actors in MSMAS allows for expanding and defining attributes of such types of system participants. We note however that MSMAS does not include details on how to do that and no examples are given. Another criterion where MSMAS does better than other methodologies is its support of *Temporal Continuity*, where MSMAS's explicit modelling of instances of agents, allows for each instance to be identified during the full system execution cycle over long time. We consider MSMAS to be slightly better than other methodologies with regard to *Concurrency* and *Human Computer Interaction* criteria, although MSMAS does not provide details on how to exploit its techniques to implement interfaces and concurrent components. None of the methodologies support explicitly the building of adaptive systems, where the system modeler can specify an agent's ability to learn or self-manage. The results of the *Supportive Feature Criteria*, show that MSMAS is better or slightly better than other methodologies on five criteria, while it is as good on the remaining three criteria. MSMAS stands out with regard to *Software and methodological support* because of the availability of the tools and although PROMETHEUS provides a supporting tool, it can verify only the models at design time while MSMAS tool allows for exporting formal models for both design time and runtime verification and validation. MSMAS

is slightly better than other methodologies because of its support for *Dynamic structure* where the methodology allows reconfigurations. This can be achieved by enabling the dynamic creation of norms and amendments to system norms, although MSMAS does not offer examples of how to do that. Another criterion where MSMAS has an advantage is in its support for the use of ontology: one of the available representations of MSMAS models is RDF, which can be expanded with ontology describing the attributes of each system component. Now let us consider the list of standard MAS development steps and concepts to highlight the strengths and weaknesses of MSMAS. All the methodologies under study are weak in respect of their support for mobile agents.

MSMAS Strengths and Limitations

Considering the full list of standard MAS development steps reveals that MSMAS is the strongest methodology because of its support for more steps than any of the other four methodologies under comparison. With regard to the first seven steps, which are related to *Problem Domain Analysis*, MSMAS includes all the steps except "Produce Sequence Diagrams", which is a step inherited from OO paradigm and in our view it is not appropriate for MAS. MSMAS includes a Roles/Activity Model which can be considered the MAS version of a Sequence Diagram. MSMAS does slightly better with regard to the identification of Goals, and Roles while only PROMETHEUS has a step to identify the system goals and only GAIA has a step to identify the roles.

The following seven steps focus on the *Agent Interaction Design*. MSMAS is as good as other methodologies and better with regard to two steps "Define Exchanged Messages" and "Specify Agent Communication Language". MSMAS has a dedicated step for the design of communication protocols and communication messages and it supports FIPA ACL as well as allowing for the definition of custom messages and protocols types. The results show a lack of steps to support "Specifying Contracts and Commitments" MSMAS approach is based on the use of norms rather than commitments, which we believe it is more suitable as explained in Chapter 4, Section 4.9. MSMAS also has no steps for "Specifying Conflict Resolution Mechanisms". Although this can be achieved through design, MSMAS does not offer ready to use patterns. We note that is a valid point for future work.

The following three steps address "Internal Agent Design", where MSMAS scored less than other methodologies such as TROPOS. This is, in our view, a lack-by-choice rather than a weakness, because MSMAS aims to be generic in these aspects rather than restricting to a particular architecture and specific concepts that could hinder the transformation of MSMAS models to another model or technology. We note however that this is an area to be evaluated and confirmed through user based evaluation as planned in our future work.

The last seven steps on the list focus on the "System and Environment Design", where MSMAS's ability to model the organisational structure is a clear advantage. The two areas we identify for the improvement of MSMAS are "Specify Agent-Environment Interaction Mechanism" and "Specify Agent Instances Location", which have to do with the lack of support for

mobile agents. Finally, the results of concept-based comparison mirrors the same conclusions we draw from the steps-based comparison.

To conclude, MSMAS introduces a new set of diagrams and notations to capture the system requirements using carefully selected concepts. This evaluation suggests that it has a very good coverage of the steps needed to model MAS from the software engineering perspective. The comparison results clearly show that MSMAS is more complete and more expressive than the other four methodologies under review.

7.4.1 Chapter Summary

In this chapter we presented a comparative study to evaluate MSMAS using the Feature Analysis Framework for Evaluating MAS Development Methodologies (MASDM). The evaluation of MSMAS models, processes and concepts is conducted by deploying benchmark measurement scales for MSMAS performance in comparison to four other existing MAS methodologies, GAIA, PROMETHEUS, TROPOS, and SONIA.

The evaluation process reveals that MSMAS successfully presents a new approach that is more complete and more expressive than other methodologies. MSMAS is particularly strong in modelling communication protocols and messages, and dynamic structures, thanks to its declarative modelling style and the use of system norms. More importantly, MSMAS support of verification at both design and run time places it ahead of other methodologies we examined. MSMAS is a comprehensive methodology because of its coverage of the full life cycle, including the implementation phase. MSMAS use of ontology also makes it more suitable for expansion and transformation from one modelling structure to another. While some methodologies focus on modelling agent internal structures and others on designing agent interaction protocols or the extended analysis, MSMAS covers all concepts and steps – more than the other methodologies under comparison – yet requires only a small number of models and steps. An area for the improvement of MSMAS is in support for mobile agents, which we consider alongside other identified limitations in the next chapter under future work.

CHAPTER 8

CONCLUSIONS

We have set the goal of developing a new multiagent software development methodology that solves the issues identified in other available methodologies and become more accessible for business developers. We achieved our goal by defining MSMAS as a complete methodology that offers an alternative approach to the building of MASs. We equipped MSMAS with features that – in our view – satisfy the commercial needs and contribute to bridging the gap between academic research activities and business users. Although MSMAS includes fewer models than some of the other methodologies, the evaluation results presented in Chapter 7 supports the view that MSMAS handles a sufficient range of MAS concepts and significantly, that it covers the entire development life cycle, which makes MSMAS more complete and expressive than many well-known methodologies. In this chapter we review our work's main aims and objectives, a summary of how these objectives were met, a summary of the contribution of this thesis and how our work resolves the identified issues. Finally, we conclude with a discussion of possible avenues for future work.

8.1 Research Objectives Revisited

Our primary goal was to develop a new MAS development methodology that improves MAS development practices and makes the MAS paradigm more accessible to business developers. We identified six objectives that collectively lead to the achievement of our goal. To address our first objective we defined a set of requirements for the building of our MAS development methodology at process level, concepts level and models level 3.1. We examine how MSMAS meets all of these requirements in the following section. The second objective was to select a suitable organisational structure for MAS and identify the necessary concepts that can support a wide range of use cases and application domains. Our chosen organisational structure is institutions and we use the concepts of normative MAS to specify the system structure, as well as to capture the system requirements. The third objective was to define a set of agent concepts

in the form of a metamodel. The MSMAS metamodel is defined with carefully selected set of concepts and is structured in a way that allows for transformation to other modelling methods. The fourth and fifth objectives were to define the development methodology (and describe it in detail) and to select a formal language in which to capture the designed system for the twin purposes of verification of the system models at design time and for validation of the implemented system.

We have presented a comprehensive methodology with straightforward steps and processes [§7.3] that cover all the stages of the development life cycle. We used formal logic to capture the system requirements and used formal methods to verify the models. The use of norms and formal validation is an enabler for a degree of self-management. The last objective was to evaluate our methodology to assess its strengths and to establish that it succeeds in meeting the objectives expressed here. The evaluation of MSMAS features in comparison to four other well-known MAS development methodologies confirms that MSMAS meets our objectives successfully.

8.2 Meeting the Objectives

Developing the MSMAS methodology turned out to have a larger scope than anticipated and some of our design decisions made during early development stages proved not to be the best at later stages. As a result, we had to revisit each model and reassess its components more than once. Although we are satisfied, thanks to the evaluation process, that MSMAS as it stands today meets its objectives, we believe there are aspects that would benefit from further development which we included in our future work plans. An overarching factor in keeping our research work structured and in consistently maintaining an ordered list of activities that led to meeting our objectives has, we believe, been our adherence to the steps defined by the Design Science Research Methodology.

We developed the MSMAS methodology based on a set of fundamental requirements that we list below:

1. **Process:** MSMAS covers the full life cycle including the implementation phase. It has a few processes with well-formed steps and offer a number of examples to help the user in exploiting its features and achieving his design goals. The inclusion of implementation phase has influenced our choice of MSMAS processes and concepts in a positive way.
2. **Concepts:** MSMAS metamodel has fewer concepts than some other metamodels, such as Hahn et al. [2009]. The reason for this was based on our view to limit our choice of agent *concepts* to those that are necessary for the building of MAS. The evaluation results of MSMAS support for MAS concepts, confirms that MSMAS supports more concepts than all other methodologies under comparison. MSMAS concepts can be mapped to other models because they fall under the set of concepts that experts in the field agree on their semantics. Furthermore, some concepts can be expanded to include

more fine-grained sub-concepts.

3. **Models:** MSMAS visual models depend on limited but sufficient notations which should make them easy to learn and produce. A number of MSMAS models are similar to well-known and commonly used models: for example, the Basic Role/Activity model is similar to the business process activity diagram, which lessens the learning curve of MSMAS for business users. MSMAS supporting tool [§E] automatically derives some models from the core ones and thereby reduces the complexity of writing their formal representation.

MSMAS attempts to capture an holistic view of the system components and their organisational structure through consideration of all aspects throughout the design process and in each model, where components of different models inform and are used in other models for mutual refinement of the system components across its visual models. MSMAS also scores equal to or higher than other MAS software development methodologies with regard to precision, accessibility, expressiveness, domain applicability, refinement, and clarity.

The MSMAS methodology gives great attention to the organisation structure through the use institutions and institutional roles. Defining the structure formally and the use of declarative system norms allow for verification of system properties during design time and the formal representation of system events allows for online monitoring.

MSMAS distinguishes itself from existing methodologies in several aspects, which we now highlight:

1. The MSMAS methodology is based on widely-used agent oriented concepts. The MSMAS metamodel is not an attempt to aggregate all concepts used in all other methods to create a unified comprehensive model that fits all, instead we focus on these concepts that we view essential to the building of MAS.
2. The MSMAS methodology covers the entire development life cycle. It has a requirements gathering phase followed by initial design then detailed design phase, and finally implementation phase, additionally, it provides formal model for design verification. Many of the existing methodologies do not cover the implementation phase, hence the lack the inclusion of concepts and techniques that support the deployment and all post-design activities. Other methodologies that include an implementation phase lack a formal model and do not support validation process hence, they are considered incomplete.
3. The MSMAS methodology scores high in terms of ease of use, as revealed by the evaluation results in Table 7.6 due to the reuse of business modelling techniques and concepts. For example the Basic Roles/Activities Model (section 4.5.5 page 124) is very similar to a business activity diagram. MSMAS has a supporting tool (Appendix E) to facilitate the design process, by means of which models can be automatically derived and generated. This support reduces the workload on the system designer, eliminates the potentially complicated task of writing formal models, which in consequence has a positive impact on the design time and helps in reducing the incidence of human errors related to

consistency.

4. The MSMAS methodology covers agent concepts at a high level and does not require a particular design pattern. We believe that the current abstraction level enables the modelling of the same complex systems in different ways as demonstrated in our case study.
5. The MSMAS methodology supports the definition of communication protocols and messages and offer an alternative to FIPA and KQML to model communication protocols using its declarative system norms. This aspect allows for the creation of a wide range of protocols and expressing them formally. The formal models of communication protocols can then be shared and observed at runtime which is a requirement for open and adaptive systems.
6. The MSMAS methodology uses the widely-supported RDF representation (in XML syntax), which given the availability of XML processing tools reduces the effort needed to map or transform a MSMAS system model into other models or to create legacy code in a particular language.
7. The MSMAS methodology supports the construction of agent organisations and offers institutions as an organisational building block. Institutions and institutional roles serve as a basis for the incorporation of advanced features into the system design such as coordination and negotiation. The organisation structure can be specified through the definition of institutional roles and their role/role norms.
8. The MSMAS methodology uses and extends ConDec^{++} to support the expressing of its defined declarative norms. MSMAS norms can, amongst other representations, be expressed in CLIMB, which allows for verifying the system models at design time. MSMAS norms can also expressed in LTL to allow of static verification, and EC that can be used in monitoring and validating system requirements.

8.3 Contribution of the Thesis and Discussion

The main contribution of this thesis is the MSMAS development methodology, MSMAS [Ch.4], that we claim combines business process modelling techniques and concepts, multiagent modelling techniques and concepts, and the principles of normative systems design in a novel and practical way. We have also developed a metamodel [Ch.3] that combines the identified concepts in a structured form and is expressed in UML. We define a number of system norms templates for four system constructs [§4.2.2]: System Goals, Institutional Roles, Communication Messages, and Business Activities, that can be used or extended to meet design requirements as needed. We expressed our defined system norms visually using ConDec^{++} and expressed their semantics formally using CLIMB (§6.3), and EC (§6.6). MSMAS models may also be described in RDF following our defined MSMAS RDF Schema (Appendix D). We have also provided a set of guidelines for designing MAS using MSMAS (§4.8) and suggested an ac-

tivity sequence (Fig.4-31) for system modelling, verification, implementation and validation using the MSMAS methodology.

We believe that our contributions will help in bridging the gap between design and implementation and help make the MAS paradigm more accessible to business users as well as researchers. Moreover our new methodology, MSMAS, addresses several problems that we have identified with existing methodologies: in the remaining part of this section, we recall the limitations of other methodologies highlighted in Section 1.1 Page 3 and discuss how we have addressed and resolved them in MSMAS:

1. *Support for Inexperienced Developers:* to respond to this issue, MSMAS combines common and widely used concepts and provides a metamodel that links business process concepts with agent concepts which helps the system designers to understand their semantics. MSMAS's use of institutions provides a ready-to-use organisational structure which eases the task of specifying the system's social aspects and encourages business users to think of their systems as social organisations, taking proper account of the human actors in business systems. The availability of a supporting tool (see Appendix E) also contributes positively to making the MAS paradigm more accessible. Furthermore, we have provided a set of guidelines to help new users get started on designing their system, in which we suggest steps in using MSMAS methodology, that are fully supported by the MSMAS designer tool.
2. *Lack of a single MAS standard:* In response to this problem, we have defined a metamodel that is limited to what we consider to be the essential concepts, where a large number of them are commonly understood across a range of other methodologies. As the evaluation results show, MSMAS is more precise and expressive than other methodologies, which means it is able to support the majority of the existing multiagent structures summarised in Appendix A. The MSMAS metamodel is not meant to be the only reference for MASs: we view it as an attempt to establish a common ground from where mapping to and from other metamodels is possible. The fact that a large number of MSMAS concepts are present in other methodologies makes it possible to transform MSMAS models to other methodologies.
3. *Absence of an holistic view:* MSMAS tries to take a balanced approach to its consideration of the various aspects of system modelling. In MSMAS, the inclusion of the organisational aspects, some cognitive aspects of agents, and norms that capture the systems requirements, contributes to building a complete picture of the system. MSMAS allows the system designer to include reactive system components such as services and non-traditional system participants such as human actors, while other methodologies miss out some of these such as human actors, reactive system participants or the specification of the environment.
4. *Lack of Whole Life-cycle Coverage:* Contrary to some existing methodologies that cover only part of the development life cycle, MSMAS covers the whole life cycle and its

models and descriptors are self-contained, the detailed description of MSMAS constructs makes it feasible to transform them to agent programming languages constructs.

5. *Gap Between Design and Implementation:* to address this issue, MSMAS is supported by a tool that can be used during the analysis, design and it exports the system specification in various formats to support implementation and deployment. The availability of the tool alongside the set of provided guidelines should help in making MSMAS attractive to commercial application developers.
6. *Absence/Presence of an Implementation Phase:* The MSMAS methodology has an explicit implementation phase. Although the current version of the supporting tool does not automatically generate legacy code, we have illustrated and provided examples of mapping MSMAS constructs to Jadex. Our future plans include implementing an automatic export of the system components into Jadex and JASON, among others.
7. *Limited Formal Representation of MAS Concepts:* to respond to the limitations of other methodologies that have no underlying formal model to support their concepts and system designs, we have created MSMAS with a formal representation of the system norms. The export function, in the MSMAS designer tool, helps in eliminating the complexity of writing system specifications, instead the user creates his/her designs with any number of system norms, then exports the formal models to verify the design and check the system properties. The exported files are ready to use with a range of freely available tools.
8. *Lack of Agreed-on MAS Development Steps:* We believe that "Industry Readiness" might be the driver for defining such steps. Although our evaluation did not include user studies to verify that MSMAS is suitable for commercial use, we believe that our approach, which combines the business process modelling with easy to understand and learn agent related concepts (See Table 7.10 and Table 7.11), is aligned with industry and closer than other methodologies to the world of commercial applications. Furthermore, MSMAS's relative simplicity should contribute to acceptance of multiagent systems in the industrial domain. The evaluation identifies several factors that should make it is easy to learn and its coverage of the full life cycle and the availability of supporting tools are all factors that can assist in the adoption of MAS. Our future work plans include user evaluation studies to confirm these conclusions.
9. *Accessibility and Industry Acceptance:* MSMAS provides a detailed description of its steps and a modelling example of a real world case study.We have used a real world use case and explained in great detail, in order to explicate the modelling process. The use of common known use case model and the novel way of linking between business processes and system goals and system participants and the reliance on known concepts and practice should make it easy to understand and to create the required models. The evaluation framework covers a broad set of software engineering concepts, steps and features and the results show that MSMAS has very good coverage across the full range.

10. *An Agent-oriented Methodology:* MSMAS is not built on top of other paradigms such as OOP, and does not use models that are inherited from other paradigms. Instead, it is built on top of MAS concepts and created from the outset for the design and development of MAS. This aims to reduce confusion with other paradigms and should enable users to think and design and develop within a MAS-oriented mental state.
11. *Organisation Structure and Security:* We recognise the importance of defining the system organisational structure, thus MSMAS gives great attention to this aspect and offers an explicit way to implement an organisational structure through the use of institutions and institutional roles. Any system participant can play one or more roles according to the specifications. Furthermore, the system designer can set a number of system norms that restrict or enforce the combination of any number of these roles. This aspect makes it possible to design and implement secure systems and even implement security policies that can be modified dynamically online during execution, but most importantly such policies can be monitored continuously for compliance.

To conclude, our aim has been to develop a comprehensive methodology to assist system designers throughout the entire software development life cycle in creating multiagent systems. We have identified issues within other methodologies and aimed to address these with our methodology. MSMAS was built based on software engineering principles such as precision, accessibility, expressiveness, and modularity. It has three phases that flow from one into another, with structured refinement steps. It allows for an iterative design process, where the system modeller can revisit the system diagrams to address any design flaws or to introduce new design decisions once an issue is discovered. MSMAS supports model derivation and traceability: its models are linked and in some cases depend quite closely on one another, in that one model may use constructs created by another, and a few models are largely generated based on previous ones, as in the case of the Composite Business Process models which are automatically generated once the user adds goals to the system goals model. We will build on this in future work to increase the range of transformations of MSMAS models to other methodologies and to target agent programming languages.

8.4 Future Work

During the course of our research, many questions and ideas arose that it was not possible to include within the scope of this work, despite their interest and attraction. These form the basis for lines of future research activities. We would also observe that the evaluation we have conducted (Ch.7) revealed some shortcomings in MSMAS that are early candidates for attention and which we detail in the first two points of the list below. This section lists ideas and topics, that we believe can improve MSMAS in particular and the engineering of MAS in general.

1. **User-Based Evaluation Study** is one of our first planned tasks. We would like to ver-

ify our assumptions with regard to MSMAS suitability for commercial user as well as academics. The feedback and results of these studies will form key input to the future development of MSMAS with regard to adding new features, improving current ones and the development of more supporting tools.

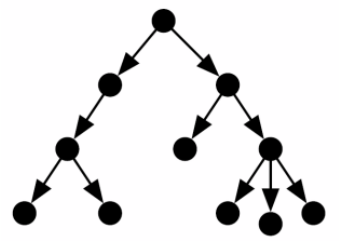
2. **Specifying Ready-to-use Features for Adaptive Systems** This is a complementary point to design patterns, but more focused on enabling the attributes of self-managing MAS. We want to include features that support the dynamic creation of norms, and modification of norms during run time according to pre-defined policy or even an emergent one. The scope of this work extends to the investigation of and the definition of conflict resolution mechanisms as well as dynamic planning.
3. **Supporting Mobile Agents and Cloud-based Systems:** The advance of cloud computing means that more businesses are moving towards the model of cloud based services and Software as a Service (SaaS) especially with the offerings of on-demand-processing power that have financial benefits for start-ups and businesses with seasonality. Support for mobile agents becomes essential, as a result. We would like to investigate new ways to specify mobile agents and how they can be deployed on virtual hosts. Which features are needed to allow migration and communications in these settings?
4. **Specifying Agent Design Patterns:** Design patterns are considered one of the most significant innovations in OOP. In MAS, the need for agent patterns is more evident due to its semi-open nature. Agent design patterns can help and speed the development process, as each pattern captures a good solution to a common problems in agent design.
5. **Dealing with General Software Development Issues** such as collecting metrics and generating logs and reports about a project, deployment methods and automated design of tests.
6. **Investigation of Norm/Norm Relations** we would like to investigate the dynamics and effects of various system norms of different types on one another. Also the definition of multiple institutions and verifying their correctness among other properties during design time.
7. **Verification of a Composition of Models** At the moment, MSMAS supports the verification of individual models. A possible future work is to investigate the static verification of model compositions to discover potential issues arising from the interaction of system norms at different levels and from different models operating concurrently.
8. **More Support for the Implementation:** We plan to add more features to the MSMAS tool including the export of MSMAS designed system directly to at least one MAS programming language. Jadex is one of the top candidates for support, followed by JASON.
9. **Building MOF model** for a full MOD-compliant metamodel we need to create an MOF model to describe MSMAS concepts and their relations.
10. **Integrate $\mathcal{S}$CIFF Reasoner with MSMAS Tool** automate the design verification process and allow for continuous verification during design time.

APPENDIX A

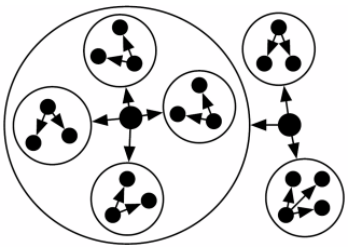
MULTIAGENT ORGANISATIONAL STRUCTURES

We have discussed briefly in Section 2.2.4 Page 17 the different organisational structures of MAS, in this appendix we summarise all structures as identified by Horling and Lesser [2005].

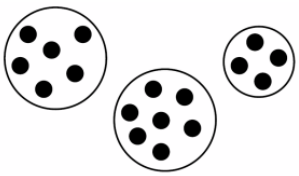
1. **Hierarchies:** Hierarchical organisation is one of the first organisation structures, and many of the earliest organisational implementations in MAS took this form. In this type Agents are arranged in a tree structure Figure A-1-A and the higher the agent in the tree the more global view it has. [Horling and Lesser, 2005]. Hierarchical organisation can be ***simple*** where a single member has the full authority of making decisions, or ***uniform*** where the authority is distributed. Hierarchical organisation structure can be very efficient due to its core notion of decomposition and its approach of divide-and-conquer which allows the system to achieve its global goals through large number of agents and through the achievements of sub-goals. An advantage of this structure is the ability of constraining the agents to a relatively small number of interactions to the total system population size. The main drawback, however, is the fragility of this structure due to the high risk of single failure point, as well as the potential slow performance with very long tree with huge number of levels.
2. **Holarchies:** Holarchies or holonic organisations are those that comprised of multi-leveled, grouped hierarchies or in other words a hierarchy of self-managed holons. The holon is a term that Arthur Koestler made of the Greek word holos (meaning "whole"), and on (meaning "part"). Koestler used that term for the first time in his 1967 book (The Ghost in the Machine) [Koestler, 1967] to describe the units that form a bigger group, they have a character derived but still distinct from the entities that are members of the same group. The independent holons coordinate their behaviour in their local environment and they remain part of the whole organisation that has supra-ordination to these holons. Each holon in this sense is composed of one or more subordinate entities (holons), and can be a member of one or more superordinate holons. Holarchies struc-



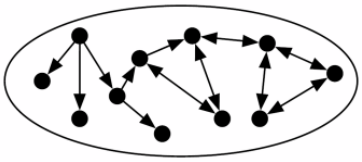
A) A hierarchical organization



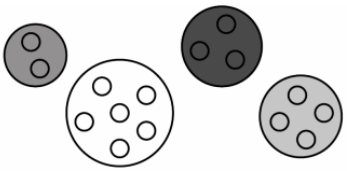
B) A holarchical organization



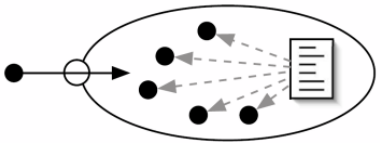
C) A coalition-based organization



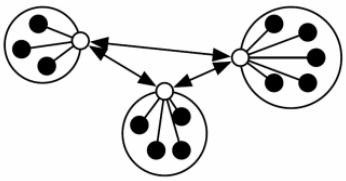
D) A team-based organization



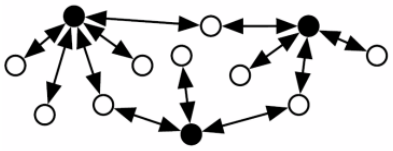
E) Congregations of agents



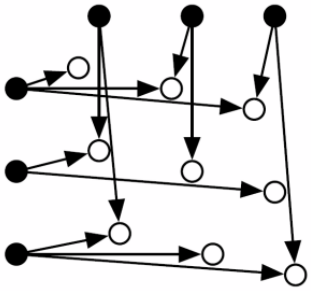
F) Societies of agents



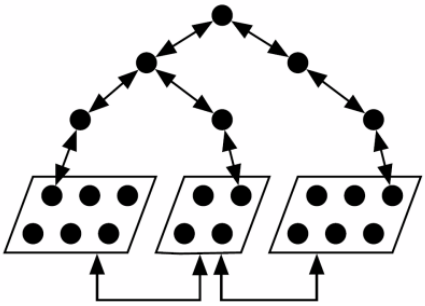
G) An agent federation



H) A multi-agent marketplace



I) A matrix organization



J) A compound organization

Figure A-1: *Organisational paradigms (used with permission [Horling and Lesser, 2005])*

ture can be used in modelling systems where goals can be recursively decomposed into subgoals and each of these subgoals to be assigned to individual agents. (Figure A-1-B). Holons has the flexibility in terms of behaviour choice that allows for coordination between both complementary and conflicting tasks. The feature in particular allows the holon adapt dynamically to new conditions. However this also can be seen as drawback, is it limits the ability to predict the systems overall performance.

3. **Coalitions:** Coalitions are generally short-term goal-directed structures, that are resolved once their purpose doesn't anymore exist. In coalition organisational structure is flat, (Figure A-1-C) and a coalition is normally treated as a single atom. Agents that form this coalition take the responsibility of coordinating their actions to serve the coalition goal. Forming a coalition can be problematic due to the complexity of presenting the goal of that coalition to its members, especially if they are self-interested agents. The advantage of this structure is potentially within the expectation of increased ability to solve goals through efficient tasks allocation between number of agents greater than a single agent. A coalition of all agents could be seen as good idea to allow access to all the system resources, however the cost of forming and maintaining this structure should be taken into account, and for every system based on its goals and the size of population there is an optimal number of agents to form coalition where the value of that coalition exceeds the cost of such coalition.
4. **Teams:** Team structure is similar to coalition as in both structures have number of agents working towards a common goal, however in the team structure the agents agreed to work together and they are trying to maximise the utility of the whole team and their individual actions are consistent with the team goal rather than their individual goals. Team members interactions might seem arbitrary (Figure A-1-D) but each agent will play one or more roles to address the sub-tasks required by the team and is expected to change roles as required over time. The team organisation structure could be seen in many systems, however only those systems that demonstrate the ability to reason precisely about their teamwork decisions are the only ones to be classified as team based organisation system. As the coalition this system benefits from maximising the problem solving capability by letting large number of agents to solve the problem rather than a single agent, the team structure over comes the high cost of forming coalition by explicitly define the team members and their clear lines and methods of communications, this however means a tighter form of coupling and less flexibility which means under unforeseen conditions can fail or stop functioning due to disability to redefine new coordination lines.
5. **Congregations:** Congregations are groups of individuals that are working together towards a specific goal, similarly to Teams and Coalition structures, but they long-lived and normally formed to achieve more a single goal and they are reason about how to facilitate the collaboration process. Figure Figure A-1-E shows that each group formed to include individuals that complement each other to offer a one or more stable capability

per the group, they are not necessary having a pre-defined single goal between them as in Coalition structure, instead they can be formed to offer the needed capabilities for some tasks. Agents forming congregations are individually rational and expected to asses the value of joining a congregation to their local goals, hence over time agents may join or leave dynamically any existing congregation. Besides the advantages of team work, congregations structure is useful to support the speed of discovery of agent partners by restricting the the population size that it needed to be searched. The issue with such structure is the reduction of quality due to the absence of rewarding and penalties for individual decommitments. In short in the organisation structure it is a trade off quality for a reduction in time, complexity or cost [Horling and Lesser, 2005].

6. **Societies:** Societies are social constructs that are naturally open, where agents can come and go, and the society remains. In this organisations agents have different goals and vary in their capabilities, while the social construct works as a common domain where they can communicate, act, and interact. The society will enforce a structure and order on its members however the interactions can remain flexible. This enforcement is normally done through imposing a set of constraint on the members behaviour (Figure A-1-F) these constraint are normally refereed to as social laws, norms or conventions. The society normally provides set of norms/rules or guidelines by which its member must act this means the overall society can function with some level of behaviour consistency. These norms however should not be seen as the ultimate mean to provide efficiency, Moses [Moses and Tennenholtz, 1995] suggests that social laws can only be used to provide a formal structure and more complex inter-agent behaviours can then be built upon that formal structure. So setting limitations and enforcing them can be used to allow the agents to have more simplified view other agents. The biggest advantage of the societies structure is the ability to establish normative behaviours monitoring mechanisms to regulate the behaviour of the society members. This can be done through monitoring their actions and then compare it to the expected patterns to identify any potential norms violations. The regulation of society can be done through explicit presentations of trust and/or reputation, or through social institutions which is a mechanism that allows agents to formalise their interactions by using contracts that can be verified by these institutions. The formation of a society requires the definition of roles, protocols, and social norms and then the determination of how the society members can join of leave the society.
7. **Federations:** In this structure the organisation members delegate some amount of their autonomy to on member who acts normally as a delegate/mediator/facilitator between this group and other groups. Each individual retain an amount of local autonomy and goals but work to a degree under the directions of the group delegate and would normally communicate only one that delegate when it requires to communicate with an external group, the delegate then present this and does the actual interaction with the external group (Figure A-1-G). Given that all interactions between external groups and this group

is done through one entity (the delegate) it makes communications easier as that delegate present a single consistent interface to the group. Thus this structure does support similar benefits to those of the holons in the Holarchies structure. The delegate might be seen as translator/transformer which receives different undirected instructions such as task requests, capability notifications and application-level data from the group member then it communicate this to the proper channel outside the group in a unified language. Given this distinguished position, the delegate might act of more functions such as perform a task allocation or monitor a progress. The drawback of this type of organisation is the possibility of delegates to become bottlenecks due to them being central points of processing and passing all group members traffic.

8. **Markets:** Markets organisation structure is an imitation real world producer-consumer system, where firstly; there is a number producers/sellers offering different kinds of services/resources, secondly; a number of buyers that compete to get hold of these services/resources by placing bids, and finally; a third type of members that run the marketplace or act as auctioneers that are responsible to collecting the bids and determining the winner (Figure A-1-H). this structure is similar to federation however the delegates (auctioneers) are just a trusted entities; they are not really presenting the members of the group.and the group members are normally have high degree of inner-group coemption. Market structure organisation normally benefits from the wealth of human economics and business research to allow for creating a sold organisation that follows the dynamics of real life marketplaces although it can be used in many domains. An example mentioned by Horling and Lesser [Horling and Lesser, 2005] is the Mariposa distributed database system [Stonebraker et al., 1996] that requires individual nodes to sell/bid on pieces of information. The use of marketplace mechanism in that example is meant o optimise the query processing process. Markets-based organisations are well situated for resources allocations and pricing if each member submit a bid of a fair presentation of its value gain, the design a successful marketplace is however a complex task and it requires full understanding of auctions't types and properties and to be able to reason about the bidding process (know as counterspeculation) and determine the auction's outcome (Know as clearing the trade) and it is NP-complete problem. These market based organisations are also vulnerable to a form od cheating know as collusion; when two or more bidders agree to reduce their bids and coordinate there behaviour.
9. **Matrices:** This is a relaxed form of the hierarchical organisation structure; so instead of the strict structure of single manger per each member of the hierarchy tree, the matrix organisation allows for many mangers to influence the directions of a single member. In this sense that structure mirrors real life situations more where a person would have number of interrelationships that come from many directions and each of them has its own objectives. Figure A-1-I present the matrix structure similar to the grid system. Matrix organisations re good specially when specifying a system where it consist of

number of limited recourses so they are normally needed to be shared also if there is a need to clearly specify how a member behaviour is influenced by multiple lines of authority. The drawback of this structure is the possibility of the some members to become dysfunctional when the managers disagree or have conflicting needs.

10. **Compounds:** With ever growing complexity many organisations might be formed from overlapped structures from the ones above. Figure A-1-J for example shows a compound organisation that is formed by combining hierarchy with a set of coalitions. This type f organisations require more sophisticated members who are able to reason under different roles depending on the conceptual level they operate on at some point of time that can change over times. From the organisation structures mentioned above some are open to a coexistence form with others as an example society structure is more likely to exist in support of other structures. Also some research has exploited the idea of a specific structure to create another one, such as the use of congregations to facilitate the dynamic formation of markets or the use of markets and hierarchies to create coalitions. These can be seen a benefits however it results into an increased level of complexity to they become harder to implement,

APPENDIX B

FIPA-AGENT COMMUNICATION LANGUAGE

Agent Communication Language is an agent is a set of agent standards created and managed by The Foundation for Intelligent Physical Agents, it allows agents to communicate using messages each one is considered a communicative act. In a communicative act, the agent performs an action normally sends a messages that are encoded in specific format. FIPA-ACL consists of three libraries as follows:

- FIPA Communicative Act Library (CAL)
- FIPA Content Language Library (CLL)
- FIPA Interaction Protocol Library (IPL)

FIPA-ACL message is formed using data based on concepts from these libraries, Figure B-1 shows the message structure. In the following section, we give more details on these libraries.

B.1 FIPA Communicative Acts Library

The FIPA Communicative Acts (CAs) Library is a catalogue of all specified generic Communicative Acts that can be applicable to any application domain. Any dialogue evolving between two agents consists of Communicative Acts blocks, that are content independent and is executed by just transferring a message from one agent to another.

Communicative acts are grouped into two categories; primitive and composed. The primitive communicative acts are those atomic actions that are created by only one communicative act. While the Composed ones are created by more than one communicative act such as when making an object of another (I *request* you to *inform*, using the composition operator ";" to sequence actions (*infom* me ;*request* from x), or using the composition operator "—" to perform a macro communicative act which is a set of possible disjunctive actions (a—b means a or b but not both). Table B.1 summaries the FIPA communicative acts [1] for full details refer to

[1] `http://www.fipa.org/specs/fipa00037/SC00037J.html`

`http://www.fipa.org/`.

Performative	Description
accept-proposal	The action of accepting a previously submitted proposal to perform an action.
agree	The action of agreeing to perform some action, possibly in the future.
cancel	The action of one agent informing another agent that the first agent no longer has the intention that the second agent performs some action.
call for proposal (cfp)	The action of calling for proposals to perform a given action.
confirm	The sender informs the receiver that a given proposition is true, where the receiver is known to be uncertain about the proposition.
disconfirm	The sender informs the receiver that a given proposition is false, where the receiver is known to believe, or believe it likely that, the proposition is true.
failure	The action of telling another agent that an action was attempted but the attempt failed.
inform	The sender informs the receiver that a given proposition is true.
inform-if	A macro action for the agent of the action to inform the recipient whether or not a proposition is true.
inform-ref	A macro action for sender to inform the receiver the object which corresponds to a descriptor, for example, a name.
not-understood	The sender of the act (for example, i) informs the receiver (for example, j) that it perceived that j performed some action, but that i did not understand what j just did. A particular common case is that i tells j that i did not understand the message that j has just sent to i.
propagate	The sender intends that the receiver treat the embedded message as sent directly to the receiver, and wants the receiver to identify the agents denoted by the given descriptor and send the received propagate message to them.
propose	The action of submitting a proposal to perform a certain action, given certain preconditions.
proxy	The sender wants the receiver to select target agents denoted by a given description and to send an embedded message to them.
query-if	The action of asking another agent whether or not a given proposition is true.

Continued on next page

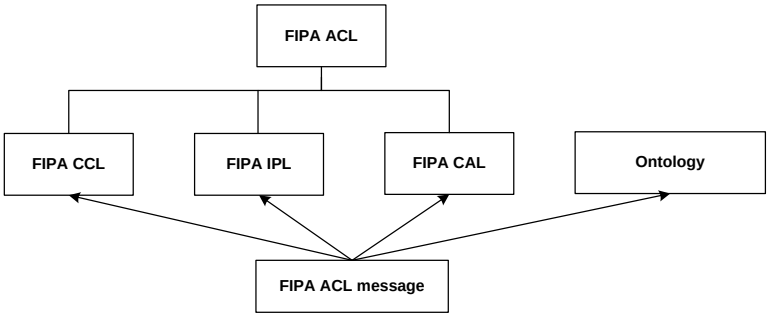


Figure B-1: *FIPA ACL Message Structure*

Table B.1 – continued from previous page

Performative	Description
query-ref	The action of asking another agent for the object referred to by a referential expression.
refuse	The action of refusing to perform a given action, and explaining the reason for the refusal.
reject-proposal	The action of rejecting a proposal to perform some action during a negotiation.
request	The sender requests the receiver to perform some action. One important class of uses of the request act is to request the receiver to perform another communicative act.
request-when	The sender wants the receiver to perform some action when some given proposition becomes true.
request-whenever	The sender wants the receiver to perform some action as soon as some proposition becomes true and thereafter each time the proposition becomes true again.
subscribe	The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.

Table B.1: *FIPA Communicative Acts Summary*

B.2 FIPA Interaction Protocols Library

Interaction protocols are predefined pattern of communication where a conversation takes place between two agents. The protocol is initiated when the "Initiator" agent send message to the receiver "Participant" agent. An example of FIPA Interaction Protocol is FIPA Request Interaction Protocol where an agent can request another to execute an action, the receiving agent might choose to agree to execute that action or reject. In case it has agreed, ti will

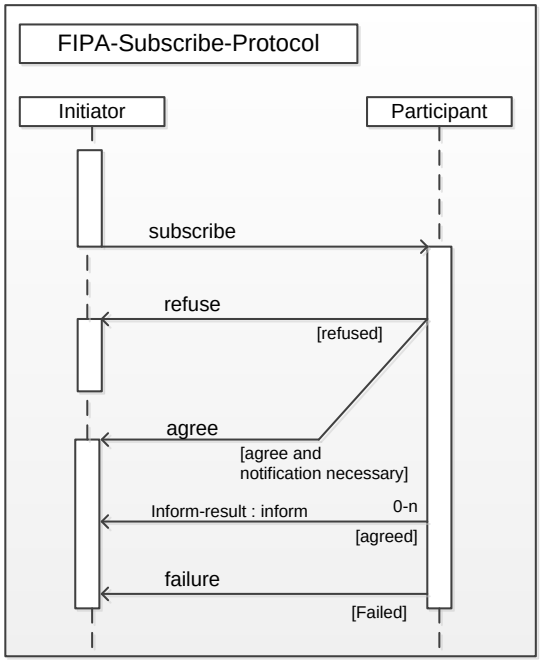


Figure B-2: *FIPA Subscribe Interaction Protocol*

inform the initiator that it is done, or failed, or done with results. FIPA provides definition and specification of the list of the following interaction protocols:

- FIPA Request Interaction Protocol.
- FIPA Query Interaction Protocol.
- FIPA Request When Interaction Protocol.
- FIPA Contract Net Interaction Protocol.
- FIPA Iterated Contract Net Interaction Protocol.
- FIPA Brokering Interaction Protocol.
- FIPA Recruiting Interaction Protocol.
- FIPA Subscribe Interaction Protocol.
- FIPA Propose Interaction Protocol.

We give more details on "FIPA Subscribe Interaction Protocol" as an example, for full details please refer to `http://www.fipa.org/`.

FIPA Subscribe Interaction Protocol:

The FIPA Subscribe Interaction Protocol allows an agent to request a receiving agent to perform an action on subscription and subsequently when the referenced object changes it would be notified. Figure B-2 is graphical presentation of this interaction protocol.

Explanation of the Protocol Flow:

The Initiator agent begins the interaction with a subscribe message that contains the reference of the objects in which they are interested. The Participant processes the subscribe message and respond with either agree or refuse action. If conditions indicated that an explicit

agreement is required, then the Participant communicates an agree in some cases the agree may be optional.

In a successful response, the Participant replies with an inform-result communication with the content being a referring expression to the subscribed objects. The Participant continues to send inform-result messages as the objects denoted by the referring expression change. If at some point after the Participant agrees, it experiences a failure, then it communicates this with a failure message, which also terminates the interaction. Otherwise, the interaction may be terminated by the Initiator using the cancel meta-protocol.

B.3 Messages in FIPA ACL

A FIPA ACL-message consists of a set of one or more message parameters. The parameters are required and defined according the message specifications. For an agent to be ACL compliant, it must:

- Be able to send and understand "not-understood" messages.
- Be able to form ACL messages correctly according to the semantic definition and specifications.
- Be able to use ACL communicative acts correctly according to their definition and specifications.
-

FIPA ACL Message Structure In any FIPA ACL Message, there is only one compulsory parameter; "the performative". Most message will contain also a sender, a receiver and a content parameters. Any agent can reply with the suitable not-understood message if it does not recognise the message received or it is not capable to administer one or more of the message parameters. Once the value can be presumed by the context of the conversation, some parameters of the message may be deleted. However, this is nor recommended practice as FIPA does not specify any means to deal with such conditions. Table B.2 summarise ACL message parameters, for more details please refer to `http://www.fipa.org/specs/fipa00061/SC00061G.html`.

Parameters	**Description**
performative	Denotes the type of the communicative act of the ACL message
sender	Denotes the identity of the sender of the message, that is, the name of the agent of the communicative act.
receiver	Denotes the identity of the intended recipients of the message.
reply-to	This parameter indicates that subsequent messages in this conversation thread are to be directed to the agent named in the reply-to parameter, instead of to the agent named in the sender parameter.

Continued on next page

Table B.2 – continued from previous page

Parameters	Description
content	Denotes the content of the message; equivalently denotes the object of the action. The meaning of the content of any ACL message is intended to be interpreted by the receiver of the message. This is particularly relevant for instance when referring to referential expressions, whose interpretation might be different for the sender and the receiver.
language	Denotes the language in which the content parameter is expressed.
encoding	Denotes the specific encoding of the content language expression.
ontology	Denotes the ontology(s) used to give a meaning to the symbols in the content expression.
protocol	Denotes the interaction protocol that the sending agent is employing with this ACL message.
conversation-id	Introduces an expression (a conversation identifier) which is used to identify the ongoing sequence of communicative acts that together form a conversation.
reply-with	Introduces an expression that will be used by the responding agent to identify this message.
in-reply-to	Denotes an expression that references an earlier action to which this message is a reply.
reply-by	Denotes a time and/or date expression which indicates the latest time by which the sending agent would like to receive a reply.

Table B.2: *FIPA ACL Message Parameters*

FIPA ACL Message Example

Example where $agent_x$ informs $agent_y$ that it accepts an offer from to send a file when the supplier is ready. $agent_y$ will inform $agent_x$ once this action happens.

(accept-proposal
:sender (agent-identifier :name agent$_x$)
:receiver (set (agent-identifier :name agent)$_y$))
:in-reply-to request12
:content
((action (agent-identifier :name agent$_y$)
(send-file DeltaUpdate.dat))
(B (agent-identifier :name agent$_y$)
(ready supplier20)))
:language FIPA-SL)

APPENDIX C

RDF, RDFS AND SPARQL

We have discussed in Chapter 4 Section 4.6 how MSMAS models can be exported as RDF files in order to support both verification and implementation processes. In this appendix we give more details on the Resource Description Framework (RDF).

C.1 Resource Description Framework (RDF)

RDF is a directed, labelled graph data format for presenting information. It is a data model that has the key concepts of resource, property, and statement. A statement is a resource-property-value triple.

RDF uses XML-based syntax to support syntactic interoperability and it is domain-independent, where the user can specify a RDF Schema to describe a specific domains.

Resources Concept

A resource as an *object* or a *thing* that we want to model, it can be "authors", "books", "cities", "stores", "web sites" ... etc. Every resource has a Uniform Resource Identifier (URI) that can be can be a Uniform Resource Locator (URL), Web address or some other kind of unique identifier.

Properties Concept

Properties are a special kind of resources used to describe relations between resources, for example "number of pages", "title", "binding" can be all properties of "books" resource. Properties in RDF are also identified by URIs (and in practice by URLs). The use of URIs to identify "things" and all relations between them enable us to have a global unique naming scheme.

Statements Concept

A statement is an object-attribute-value triple, consisting of a resource, a property, and a value. Values can either be resources or literals. Literals are atomic values (strings).

An example of a statement is:

UploadFile is sub goal of *UpdateMarketplaceOffers*

The simplest way of interpreting this statement is to use the definition and consider the triple:

```
(UpdateMarketplaceOffers,
http://www.mydomain.org/hasSubGoal, #UploadFile)
```

We can think of this triple (x, P, y) as a logical formula P(x, y), where the binary predicate P relates the object x to the object y.

C.2 RDF Schema

RDF Schema is a primitive ontology language that defines the vocabulary used in RDF data models. Users use RDFS to define the vocabulary, to specify which properties apply to which kinds of objects and what values they can take, and to describe the relationships between objects. The key concepts of RDF Schema are *class*, *subclass* relations, *property*, *subproperty* relations, and *domain* and *range* restrictions. For example:

CompositeGoal is subclass of SystemGoal

This sentence means that all CompositeGolas are also SystemGoals. RDF/RDFS enables us to model particular domains so fixing the semantics of certain piece of knowledge to be respected across all software that uses this knowledge.

C.3 SPARQL Query Language

The SPARQL Query Language[1] is a W3C Recommendation for querying RDF. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs.

SPARQL queries are similar to SQL queries; they have a SELECT-FROM-WHERE structure:

- *SELECT* specifies the projection: the number and order of retrieved data.
- *FROM* is used to specify the source being queried. This clause is optional; when it is not specified, we can simply assume we are querying the knowledge base of a particular system.
- *WHERE* imposes constraints on possible solutions in the form of graph pattern templates and boolean constraints.

An example query below, will return the names of all System Goals in an MSMAS model:

```
SELECT ?x ?y WHERE {
```

[1] URL: `http://www.w3.org/TR/rdf-sparql-query/`

```
?SystemGoal:hasName ?y }
```

C.4 Further Readings

- A Semantic Web Primer [Antoniou, 2004].
- W3C Semantic Web Activity (`http://www.w3.org/2001/sw/`).
- Resource Description Framework (RDF) (`http://www.w3.org/RDF/`).

APPENDIX D

MSMAS RDF SCHEMA

We have presented in Chapter 4 Section 4.6 an excerpt of MSMAS RDF Schema, in this appendix we include the Schema in full.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [
    <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY a 'http://protege.stanford.edu/system#'>
    <!ENTITY kb 'http://protege.stanford.edu/kb#'>
    <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
]>
<rdf:RDF xmlns:rdf="&rdf;"
    xmlns:a="&a;"
    xmlns:kb="&kb;"
    xmlns:rdfs="&rdfs;">
<rdfs:Class rdf:about="&kb;ActivityRelation"
    a:role="abstract"
    rdfs:label="ActivityRelation">
   <rdfs:subClassOf rdf:resource="&kb;Relationship"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Actor"
    rdfs:label="Actor">
   <rdfs:subClassOf rdf:resource="&kb;ProactiveSystemParticipant"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Agent"
    rdfs:label="Agent">
   <rdfs:subClassOf rdf:resource="&kb;ProactiveSystemParticipant"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;BasicBusinessProcess"
    rdfs:label="BasicBusinessProcess">
   <rdfs:subClassOf rdf:resource="&kb;BusinessProcess"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;BasicBusinessProcessModel"
```

```
    rdfs:label="BasicBusinessProcessModel">
   <rdfs:subClassOf rdf:resource="&kb;BusinessProcessModels"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;BasicSystemGoal"
    rdfs:label="BasicSystemGoal">
   <rdfs:subClassOf rdf:resource="&kb;SystemGoal"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Belief"
    rdfs:label="Belief">
   <rdfs:subClassOf rdf:resource="&kb;BeliefBase"/>
   <rdfs:subClassOf rdf:resource="&kb;CommunicationComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;BeliefBase"
    rdfs:label="BeliefBase">
   <rdfs:subClassOf rdf:resource="&kb;SystemComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;BusinessActivity"
    rdfs:label="BusinessActivity">
   <rdfs:subClassOf rdf:resource="&kb;BasicBusinessProcessModel"/>
   <rdfs:subClassOf rdf:resource="&kb;BusinessEntity"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;BusinessEntity"
    a:role="abstract"
    rdfs:label="BusinessEntity">
   <rdfs:subClassOf rdf:resource="&kb;SystemComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;BusinessProcess"
    a:role="abstract"
    rdfs:label="BusinessProcess">
   <rdfs:subClassOf rdf:resource="&kb;BusinessEntity"/>
   <rdfs:subClassOf rdf:resource="&kb;SpecificBusinessProcessModel"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;BusinessProcessModels"
    rdfs:label="BusinessProcessModels">
   <rdfs:subClassOf rdf:resource="&kb;MultiAgentSystem"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;BusinessProcessRelation"
    a:role="abstract"
    rdfs:label="BusinessProcessRelation">
   <rdfs:subClassOf rdf:resource="&kb;Relationship"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Capability"
    rdfs:label="Capability">
   <rdfs:subClassOf rdf:resource="&kb;SystemComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;CommunicationComponents"
    a:role="abstract"
    rdfs:label="CommunicationComponents">
```

```
   <rdfs:subClassOf rdf:resource="&kb;MultiAgentSystem"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;ConceptualSystemPlan"
    rdfs:label="ConceptualSystemPlan">
   <rdfs:subClassOf rdf:resource="&kb;SystemPlan"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;DeclarativeNorms"
    rdfs:label="DeclarativeNorms">
   <rdfs:subClassOf rdf:resource="&kb;SystemComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;E-InstitutionGovernor"
    a:role="abstract"
    rdfs:label="E-InstitutionGovernor">
   <rdfs:subClassOf rdf:resource="&kb;InstitutionMember"/>
</rdfs:Class>
<rdf:Property rdf:about="&kb;E-InstitutionMemberRole"
    rdfs:label="E-InstitutionMemberRole">
   <rdfs:domain rdf:resource="&kb;InstitutionMember"/>
   <rdfs:range rdf:resource="&kb;InstitutionRole"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;E-InstitutionMemberType"
    a:maxCardinality="1"
    a:range="cls"
    rdfs:label="E-InstitutionMemberType">
   <rdfs:domain rdf:resource="&kb;InstitutionMember"/>
   <a:allowedParents rdf:resource="&kb;ProactiveSystemParticipant"/>
   <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;E-InstitutionNormsType"
    a:maxCardinality="1"
    a:range="cls"
    rdfs:label="E-InstitutionNormsType">
   <rdfs:domain rdf:resource="&kb;InstitutionNorms"/>
   <a:allowedParents rdf:resource="&kb;SystemParticipantNorms"/>
   <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>
<rdfs:Class rdf:about="&kb;ExecutableSystemPlan"
    rdfs:label="ExecutableSystemPlan">
   <rdfs:subClassOf rdf:resource="&kb;SystemPlan"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;GeneralSystemGoal"
    rdfs:label="GeneralSystemGoal">
   <rdfs:subClassOf rdf:resource="&kb;SystemGoal"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Goal"
    a:role="abstract"
    rdfs:label="Goal">
   <rdfs:subClassOf rdf:resource="&kb;SystemComponents"/>
```

```
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Institution"
    rdfs:label="Institution">
   <rdfs:subClassOf rdf:resource="&kb;MultiAgentSystem"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;InstitutionMember"
    a:role="abstract"
    rdfs:label="InstitutionMember">
   <rdfs:subClassOf rdf:resource="&kb;Institution"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;InstitutionNorms"
    rdfs:label="InstitutionNorms">
   <rdfs:subClassOf rdf:resource="&kb;Institution"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;InstitutionRole"
    rdfs:label="InstitutionRole">
   <rdfs:subClassOf rdf:resource="&kb;Institution"/>
</rdfs:Class>
<rdf:Property rdf:about="&kb;MSMAS_Class32"
    a:maxCardinality="1"
    rdfs:label="MSMAS_Class32">
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdfs:Class rdf:about="&kb;Message"
    rdfs:label="Message">
   <rdfs:subClassOf rdf:resource="&kb;Protocol"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;MultiAgentSystem"
    rdfs:label="MultiAgentSystem">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Norm"
    a:role="abstract"
    rdfs:label="Norm">
   <rdfs:subClassOf rdf:resource="&kb;SystemComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Plan"
    a:role="abstract"
    rdfs:label="Plan">
   <rdfs:subClassOf rdf:resource="&kb;CommunicationComponents"/>
   <rdfs:subClassOf rdf:resource="&kb;SystemComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;ProactiveSystemParticipant"
    a:role="abstract"
    rdfs:label="ProactiveSystemParticipant">
   <rdfs:subClassOf rdf:resource="&kb;SystemParticipants"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Protocol"
```

```
    rdfs:label="Protocol">
   <rdfs:subClassOf rdf:resource="&kb;CommunicationComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;ReactiveSystemParticipant"
    a:role="abstract"
    rdfs:label="ReactiveSystemParticipant">
   <rdfs:subClassOf rdf:resource="&kb;SystemParticipants"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Relationship"
    a:role="abstract"
    rdfs:label="Relationship">
   <rdfs:subClassOf rdf:resource="&kb;SystemComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Role"
    rdfs:label="Role">
   <rdfs:subClassOf rdf:resource="&kb;SystemComponents"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Service"
    rdfs:label="Service">
   <rdfs:subClassOf rdf:resource="&kb;ReactiveSystemParticipant"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SpecificBusinessProcess"
    rdfs:label="SpecificBusinessProcess">
   <rdfs:subClassOf rdf:resource="&kb;BusinessProcess"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SpecificBusinessProcessModel"
    rdfs:label="SpecificBusinessProcessModel">
   <rdfs:subClassOf rdf:resource="&kb;BusinessProcessModels"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SpecificSystemGoal"
    rdfs:label="SpecificSystemGoal">
   <rdfs:subClassOf rdf:resource="&kb;SystemGoal"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Step"
    rdfs:label="Step">
   <rdfs:subClassOf rdf:resource="&kb;Plan"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemComponents"
    a:role="abstract"
    rdfs:label="SystemComponents">
   <rdfs:subClassOf rdf:resource="&kb;MultiAgentSystem"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemEnvironment"
    rdfs:label="SystemEnvironment">
   <a:_slot_constraints rdf:resource="&kb;KB_418084_Class20"/>
   <rdfs:subClassOf rdf:resource="&kb;SystemParticipants"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemGoal"
```

```
    a:role="abstract"
    rdfs:label="SystemGoal">
   <rdfs:subClassOf rdf:resource="&kb;Goal"/>
   <rdfs:subClassOf rdf:resource="&kb;SystemGoalsModel"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemGoalRelation"
    rdfs:label="SystemGoalRelation">
   <a:_slot_constraints rdf:resource="&kb;MSMAS_Class29"/>
   <rdfs:subClassOf rdf:resource="&kb;Relationship"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemGoalsModel"
    rdfs:label="SystemGoalsModel">
   <rdfs:subClassOf rdf:resource="&kb;MultiAgentSystem"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemNorms"
    rdfs:label="SystemNorms">
   <rdfs:subClassOf rdf:resource="&kb;Norm"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemParticipantNorms"
    rdfs:label="SystemParticipantNorms">
   <rdfs:subClassOf rdf:resource="&kb;Norm"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemParticipantPlan"
    rdfs:label="SystemParticipantPlan">
   <rdfs:subClassOf rdf:resource="&kb;Plan"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemParticipants"
    rdfs:label="SystemParticipants">
   <rdfs:subClassOf rdf:resource="&kb;SystemParticipantsModels"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemParticipantsModels"
    rdfs:label="SystemParticipantsModels">
   <rdfs:subClassOf rdf:resource="&kb;MultiAgentSystem"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemPlan"
    a:role="abstract"
    rdfs:label="SystemPlan">
   <rdfs:subClassOf rdf:resource="&kb;Plan"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;UseCase"
    rdfs:label="UseCase">
   <rdfs:subClassOf rdf:resource="&kb;UseCaseModels"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;UseCaseModels"
    rdfs:label="UseCaseModels">
   <rdfs:subClassOf rdf:resource="&kb;MultiAgentSystem"/>
</rdfs:Class>
<rdf:Property rdf:about="&kb;achievedBy"
```

```
    a:minCardinality="1"
    rdfs:label="achievedBy">
   <rdfs:range rdf:resource="&kb;BusinessProcess"/>
   <rdfs:domain rdf:resource="&kb;SystemGoal"/>
   <a:inverseProperty rdf:resource="&kb;achievingGoal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;achievingGoal"
    a:maxCardinality="1"
    rdfs:label="achievingGoal">
   <a:allowedClasses rdf:resource="&kb;BasicSystemGoal"/>
   <rdfs:domain rdf:resource="&kb;BusinessProcess"/>
   <a:allowedClasses rdf:resource="&kb;SpecificSystemGoal"/>
   <rdfs:range rdf:resource="&kb;SystemGoal"/>
   <a:inverseProperty rdf:resource="&kb;achievedBy"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;associatedRole"
    a:minCardinality="1"
    rdfs:label="associatedRole">
   <rdfs:domain rdf:resource="&kb;Capability"/>
   <rdfs:range rdf:resource="&kb;Role"/>
   <a:inverseProperty rdf:resource="&kb;hasCapability"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;businessActivityId"
    a:maxCardinality="1"
    a:minCardinality="1"
    a:range="integer"
    rdfs:label="businessActivityId">
   <rdfs:domain rdf:resource="&kb;BusinessActivity"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;businessActivityName"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="businessActivityName">
   <rdfs:domain rdf:resource="&kb;BusinessActivity"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;businessActivityParticipants"
    a:minCardinality="1"
    rdfs:label="businessActivityParticipants">
   <a:allowedClasses rdf:resource="&kb;Actor"/>
   <a:allowedClasses rdf:resource="&kb;Agent"/>
   <rdfs:domain rdf:resource="&kb;BusinessActivity"/>
   <a:allowedClasses rdf:resource="&kb;Service"/>
   <a:allowedClasses rdf:resource="&kb;SystemEnvironment"/>
   <rdfs:range rdf:resource="&kb;SystemParticipants"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;businessProcessId"
```

```
    a:maxCardinality="1"
    a:minCardinality="1"
    a:range="integer"
    rdfs:label="businessProcessId">
   <rdfs:domain rdf:resource="&kb;BusinessProcess"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;businessProcessName"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="businessProcessName">
   <rdfs:domain rdf:resource="&kb;BusinessProcess"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;businessProcessType"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="businessProcessType">
   <a:allowedClasses rdf:resource="&kb;BasicBusinessProcess"/>
   <rdfs:range rdf:resource="&kb;BusinessProcess"/>
   <rdfs:domain rdf:resource="&kb;BusinessProcess"/>
   <a:allowedClasses rdf:resource="&kb;SpecificBusinessProcess"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;description"
    a:maxCardinality="1"
    rdfs:label="description">
   <rdfs:domain rdf:resource="&kb;MultiAgentSystem"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;eInstituationName"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="eInstituationName">
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;error"
    rdfs:label="error">
   <rdfs:domain rdf:resource="&kb;MultiAgentSystem"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;goalId"
    a:maxCardinality="1"
    a:range="integer"
    rdfs:label="goalId">
   <rdfs:domain rdf:resource="&kb;Goal"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;goalName"
```

```
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="goalName">
   <rdfs:domain rdf:resource="&kb;Goal"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;goalType"
    a:maxCardinality="1"
    rdfs:label="goalType">
   <a:allowedClasses rdf:resource="&kb;BasicSystemGoal"/>
   <a:allowedClasses rdf:resource="&kb;GeneralSystemGoal"/>
   <rdfs:domain rdf:resource="&kb;Goal"/>
   <a:allowedClasses rdf:resource="&kb;SpecificSystemGoal"/>
   <rdfs:range rdf:resource="&kb;SystemGoal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasBusinessNorm"
    rdfs:label="hasBusinessNorm">
   <rdfs:domain rdf:resource="&kb;BusinessEntity"/>
   <rdfs:range rdf:resource="&kb;SystemNorms"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasCapability"
    a:minCardinality="1"
    rdfs:label="hasCapability">
   <rdfs:range rdf:resource="&kb;Capability"/>
   <rdfs:domain rdf:resource="&kb;Role"/>
   <a:inverseProperty rdf:resource="&kb;associatedRole"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSubBusinessProcess"
    rdfs:label="hasSubBusinessProcess">
   <rdfs:range rdf:resource="&kb;BusinessProcess"/>
   <rdfs:domain rdf:resource="&kb;SpecificBusinessProcess"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSubGoal"
    a:minCardinality="1"
    rdfs:label="hasSubGoal">
   <a:allowedClasses rdf:resource="&kb;BasicSystemGoal"/>
   <rdfs:domain rdf:resource="&kb;GeneralSystemGoal"/>
   <a:allowedClasses rdf:resource="&kb;SpecificSystemGoal"/>
   <rdfs:domain rdf:resource="&kb;SpecificSystemGoal"/>
   <rdfs:range rdf:resource="&kb;SystemGoal"/>
   <a:inverseProperty rdf:resource="&kb;hasSuperGoal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSubPlan"
    a:minCardinality="1"
    rdfs:label="hasSubPlan">
   <rdfs:domain rdf:resource="&kb;ConceptualSystemPlan"/>
   <a:allowedClasses rdf:resource="&kb;ConceptualSystemPlan"/>
   <a:allowedClasses rdf:resource="&kb;ExecutableSystemPlan"/>
```

```
   <rdfs:range rdf:resource="&kb;SystemPlan"/>
   <a:inverseProperty rdf:resource="&kb;hasSuperPlan"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSuperBasicBusinessProcess"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="hasSuperBasicBusinessProcess">
   <rdfs:range rdf:resource="&kb;BasicBusinessProcess"/>
   <rdfs:domain rdf:resource="&kb;BusinessActivity"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSuperBusinessProcess"
    a:maxCardinality="1"
    rdfs:label="hasSuperBusinessProcess">
   <rdfs:domain rdf:resource="&kb;BasicBusinessProcess"/>
   <rdfs:range rdf:resource="&kb;SpecificBusinessProcess"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSuperGoal"
    a:minCardinality="1"
    rdfs:label="hasSuperGoal">
   <rdfs:domain rdf:resource="&kb;BasicSystemGoal"/>
   <a:allowedClasses rdf:resource="&kb;GeneralSystemGoal"/>
   <a:allowedClasses rdf:resource="&kb;SpecificSystemGoal"/>
   <rdfs:domain rdf:resource="&kb;SpecificSystemGoal"/>
   <rdfs:range rdf:resource="&kb;SystemGoal"/>
   <a:inverseProperty rdf:resource="&kb;hasSubGoal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSuperPlan"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="hasSuperPlan">
   <rdfs:range rdf:resource="&kb;ConceptualSystemPlan"/>
   <rdfs:domain rdf:resource="&kb;ExecutableSystemPlan"/>
   <a:inverseProperty rdf:resource="&kb;hasSubPlan"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;isA"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="isA">
   <rdfs:domain rdf:resource="&kb;Role"/>
   <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;isResponsibilityOf"
    a:minCardinality="1"
    rdfs:label="isResponsibilityOf">
   <rdfs:domain rdf:resource="&kb;SystemGoal"/>
   <rdfs:range rdf:resource="&kb;SystemParticipants"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;isStep"
```

```
    a:maxCardinality="1"
    rdfs:label="isStep">
   <rdfs:domain rdf:resource="&kb;BusinessActivity"/>
   <rdfs:range rdf:resource="&kb;Step"/>
   <a:inverseProperty rdf:resource="&kb;stepIsequvilentToAtivity"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;modelId"
    a:maxCardinality="1"
    a:minCardinality="1"
    a:range="integer"
    rdfs:label="modelId">
   <rdfs:domain rdf:resource="&kb;BusinessProcessModels"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;modelName"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="modelName">
   <rdfs:domain rdf:resource="&kb;BusinessProcessModels"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;planId"
    a:maxCardinality="1"
    a:minCardinality="1"
    a:range="integer"
    rdfs:label="planId">
   <rdfs:domain rdf:resource="&kb;Plan"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;planInitPreCondition"
    a:maxCardinality="1"
    rdfs:label="planInitPreCondition">
   <a:allowedClasses rdf:resource="&kb;Belief"/>
   <a:allowedClasses rdf:resource="&kb;Norm"/>
   <rdfs:domain rdf:resource="&kb;Plan"/>
   <rdfs:range rdf:resource="&kb;SystemComponents"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;planName"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="planName">
   <rdfs:domain rdf:resource="&kb;Plan"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;planPostCondition"
    a:maxCardinality="1"
    rdfs:label="planPostCondition">
   <rdfs:range rdf:resource="&kb;Belief"/>
```

```
   <rdfs:domain rdf:resource="&kb;Plan"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;postCondition"
    rdfs:label="postCondition">
   <rdfs:range rdf:resource="&kb;Belief"/>
   <rdfs:domain rdf:resource="&kb;BusinessEntity"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;preCondition"
    rdfs:label="preCondition">
   <a:allowedClasses rdf:resource="&kb;Belief"/>
   <rdfs:domain rdf:resource="&kb;BusinessEntity"/>
   <a:allowedClasses rdf:resource="&kb;Norm"/>
   <rdfs:range rdf:resource="&kb;SystemComponents"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;relationshipId"
    a:maxCardinality="1"
    a:minCardinality="1"
    a:range="integer"
    rdfs:label="relationshipId">
   <rdfs:domain rdf:resource="&kb;Relationship"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;relationshipName"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="relationshipName">
   <rdfs:domain rdf:resource="&kb;Relationship"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;representedByConceptualPlan"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="representedByConceptualPlan">
   <rdfs:range rdf:resource="&kb;ConceptualSystemPlan"/>
   <rdfs:domain rdf:resource="&kb;SpecificBusinessProcess"/>
   <a:inverseProperty rdf:resource="&kb;representingBusinessProcess"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;representedByExecutablePlan"
    a:minCardinality="1"
    rdfs:label="representedByExecutablePlan">
   <rdfs:domain rdf:resource="&kb;BasicBusinessProcess"/>
   <rdfs:range rdf:resource="&kb;ExecutableSystemPlan"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;representingBusinessProcess"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="representingBusinessProcess">
   <rdfs:domain rdf:resource="&kb;ConceptualSystemPlan"/>
```

```
   <rdfs:range rdf:resource="&kb;SpecificBusinessProcess"/>
   <a:inverseProperty rdf:resource="&kb;representedByConceptualPlan"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;stepId"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="stepId">
   <rdfs:domain rdf:resource="&kb;Step"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;stepIsequvilentToAtivity"
    a:maxCardinality="1"
    rdfs:label="stepIsequvilentToAtivity">
   <rdfs:range rdf:resource="&kb;BusinessActivity"/>
   <rdfs:domain rdf:resource="&kb;Step"/>
   <a:inverseProperty rdf:resource="&kb;isStep"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;stepName"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="stepName">
   <rdfs:domain rdf:resource="&kb;Step"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;systemPlanId"
    a:maxCardinality="1"
    a:range="integer"
    rdfs:label="systemPlanId">
   <rdfs:domain rdf:resource="&kb;SystemPlan"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;systemPlanName"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="systemPlanName">
   <rdfs:domain rdf:resource="&kb;SystemPlan"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;systemPlanType"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="systemPlanType">
   <a:allowedClasses rdf:resource="&kb;ConceptualSystemPlan"/>
   <a:allowedClasses rdf:resource="&kb;ExecutableSystemPlan"/>
   <rdfs:domain rdf:resource="&kb;SystemPlan"/>
   <rdfs:range rdf:resource="&kb;SystemPlan"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;useCaseBusinessProcess"
```

```
    a:minCardinality="1"
    rdfs:label="useCaseBusinessProcess">
   <rdfs:range rdf:resource="&kb;BusinessProcess"/>
   <rdfs:domain rdf:resource="&kb;UseCase"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;useCaseId"
    a:maxCardinality="1"
    a:minCardinality="1"
    a:range="integer"
    rdfs:label="useCaseId">
   <rdfs:domain rdf:resource="&kb;UseCaseModels"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;useCaseName"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="useCaseName">
   <rdfs:domain rdf:resource="&kb;UseCaseModels"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;useCaseSystemParticipant"
    rdfs:label="useCaseSystemParticipant">
   <rdfs:range rdf:resource="&kb;SystemParticipants"/>
   <rdfs:domain rdf:resource="&kb;UseCase"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;verified"
    a:maxCardinality="1"
    a:range="boolean"
    rdfs:label="verified">
   <rdfs:domain rdf:resource="&kb;MultiAgentSystem"/>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:RDF>
```

APPENDIX E

MSMAS DESIGNER TOOL

In support of MSMAS methodology we have developed the MSMAS Designer Tool (MSMASDT). A tool that allows the system designers to produce the full set of visual models and descriptors of their systems. It also enables them to export the formal models in various formats. We plan to make the tool available to download from `http://www.elakehal.com/msmas` with the user manual and some example projects. MSMASDT has an intuitive user interface E-1 which makes it easy to use. The main functions of MSMASDT at the project level are:

- Create a new project, or open an existing project.
- Save project, and Save a copy of the project.
- Export the system model as RDF.
- Export the system formal model as LTL.
- Export the system formal model as CLIMB.
- Export current visual model as an image.

While the main functions of MSMASDT at model level are:

- Create a new model.
- Edit the model properties.
- Add model component/norm.
- Edit a component/norm.
- delete a component/norm.

When the user trying to add a new component, he is asked to complete the relevant properties. The component/model description and properties are then saved as a descriptor and form a part of the RDF model. The tool helps also in keeping the task of naming system components consistent by appending a prefix that indicates the type of each system component; for example when the designer adds a composite system goal called "ManageShoppingBasket" the tool changes it to "CSG_ManageShoppingBasket". This helps the designer to recognise the design artifacts and the types of components during design time, making the design tool a great help

in simplifying and maintaining naming consistency.

The RDF/xml file contains all entities and their relationships which define the system structure as specified by MSMAS. All models can be edited directly/inderictly through the tool graphical interface. Figure E-1 shows a screen shot of MSMASDT with arrows highlighting its panels.

In this appendix we show a selection of screen shots taken of MSMAS Designer Tool (MSMASDT).

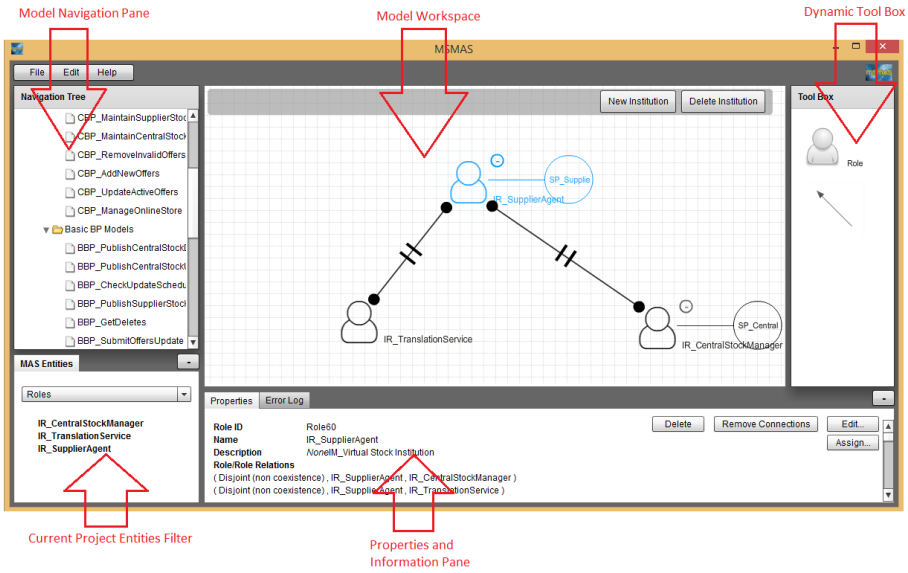


Figure E-1: *Screen shot of MSMASDT showing its interface components*

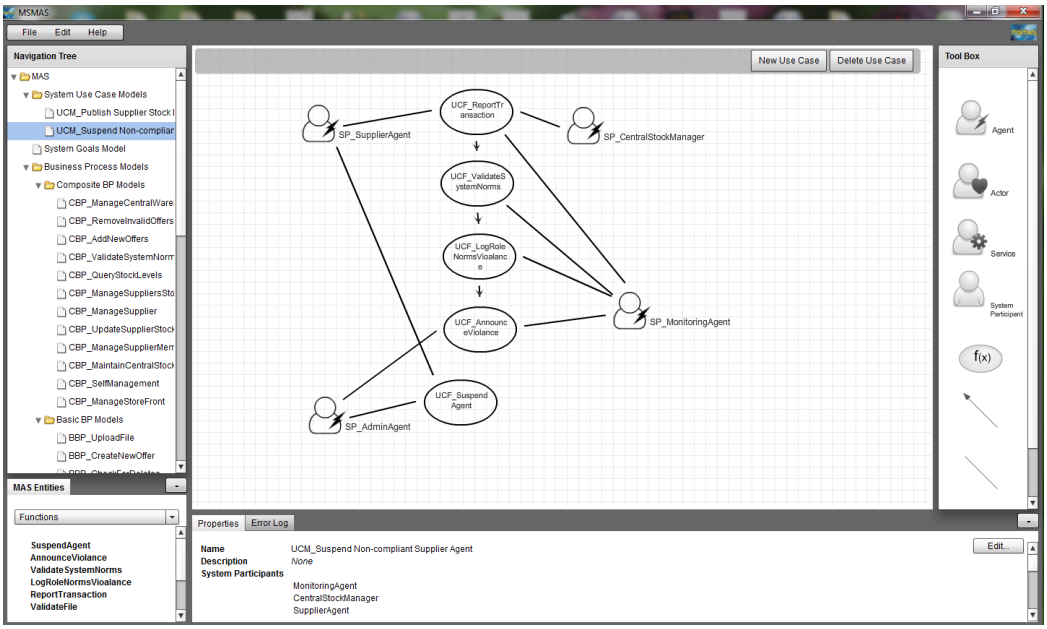


Figure E-2: *Screen shot of MSMASDT showing a use case model*

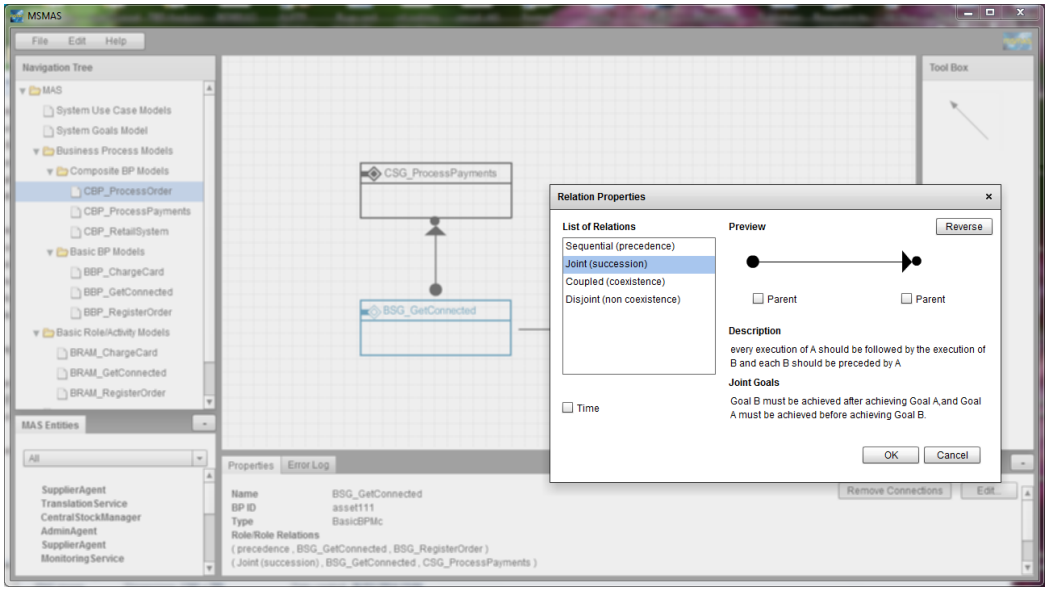


Figure E-3: *Screen shot of MSMASDT showing Goal/Goal Relation Selector to Specify a goal/goal norm*

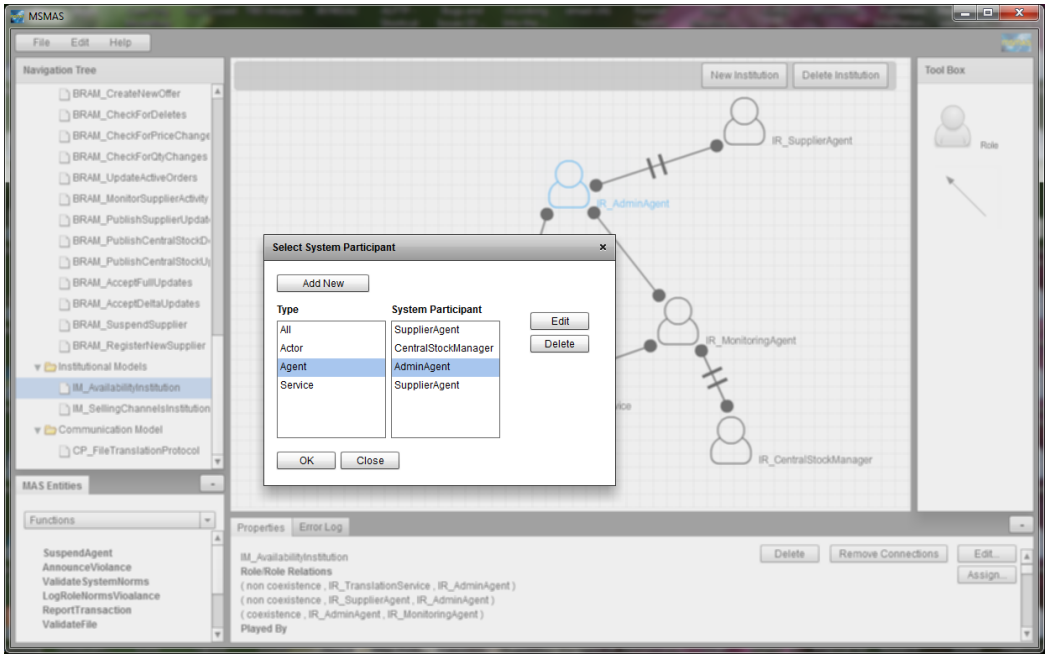


Figure E-4: *Screen shot of MSMASDT showing how to assign a system participant and Institutional Role during the design of a System Institutions*

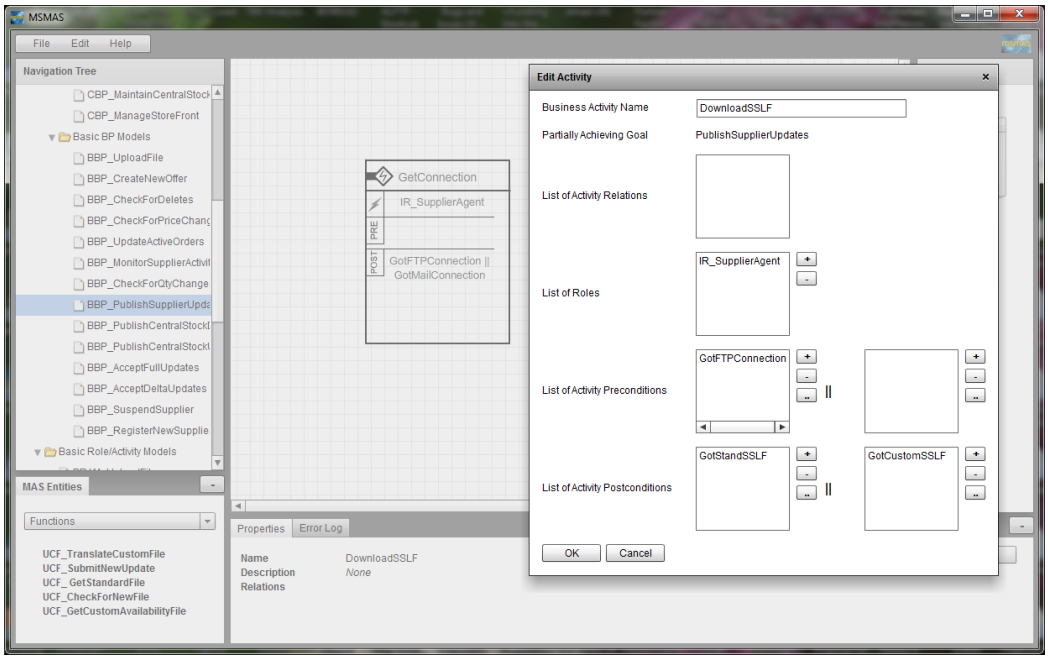


Figure E-5: *Screen shot of MSMASDT showing adding/editing a business activity to the Basic Roles/Activities Model*

APPENDIX F

FULL LIST OF CONDEC CONSTRAINTS AND RELATIONS

We include in this appendix the full list of ConDec constraints, refer to [Pesic et al., 2007, Pesic and van der Aalst, 2006a] and [van der Aalst and Pesic, 2006] for full details.

Figure F-1: *Existence Formulas and their Notation, [van der Aalst and Pesic, 2006]*

II) RELATION BETWEEN EVENTS FORMULAS

1. RESPONDED EXISTENCE formula existence_A_response_B(A: activity, B: activity)	(existence_A(A) -> existenceA(B));	
2. CO-EXISTENCE formula co_existence_A_and_B(A: activity, B: activity)	(existence(A) <-> existence(B));	
3. RESPONSE formula A_response_B(A: activity, B: activity)	□((activity == A -> existence(B)));	
4. PRECEDENCE formula A_precedence_B(A: activity, B: activity)	(existence_A(B) -> (!(activity == B) _U activity == A));	
5. SUCCESSION formula A_succession_B(A: activity, B: activity)	(A_response_B(A,B) ∧ A_precedence_B(A,B));	
6. ALTERNATE RESPONSE formula A_alternate_response_B(A: activity, B: activity)	(A_response_B(A,B) ∧ B_always_between_A(A,B)*);	
7. ALTERNATE PRECEDENCE formula A_alternate_precedence_B(A: activity, B: activity)	(A_precedence_B(A,B) ∧ B_always_between_A(B,A)*);	
8. ALTERNATE SUCCESSION formula A_alternate_succession_B(A: activity, B: activity)	(A_alternate_precedence_B(A,B) ∧ A_alternate_response_B(A,B));	
9. CHAIN RESPONSE formula chain_A_response_B(A: activity, B: activity)	A_response_B(A,B) ∧ □((activity == A -> _O(activity == B)));	
10. CHAIN PRECEDENCE formula chain_A_precedence_B(A: activity, B: activity)	(A_precedence_B(A,B) ∧ □((_O(activity == B) -> activity == A)));	
11. CHAIN SUCCESSION formula chain_A_succession_B(A: activity, B: activity)	(chain_A_response_B(A,B) ∧ chain_A_precedence_B(A,B));	
* subformula B_always_between_A(A: activity, B: activity)	□((activity == A -> _O(A_precedence_B(B,A))));	

Figure F-2: *Relations Formulas and their Notation, [van der Aalst and Pesic, 2006]*

III) NEGATION RELATION BETWEEN EVENTS FORMULAS

12.a. RESPONDED ABSENCE formula existence_A_response_notB(A: activity, B: activity)	(existence_A(A) -> absence(B));	
12.b. NOT CO_EXISTENCE formula existence_A_response_notB(A: activity, B: activity)	(existence_A_response_notB(A,B) ∧ existence_A_response_notB(B,A));	
13.a. NEGATION RESPONSE formula A_response_notB(A: activity, B: activity)	□((activity == A -> absence(B)));	
13.b. NEGATION PRECEDENCE formula notA_precedence_B(A: activity, B: activity)	□((existence(B) -> activity != A));	
13.c. NEGATION SUCCESSION formula notA_succession_notB(A: activity, B: activity)	(A_response_notB(A,B) ∧ notA_precedence_B(A,B));	
14. NEGATION ALTERNATE REPONSE formula A_not_alternate_response_B(A: activity, B: activity)	B_never_between_A(A,B)**;	
15. NEGATION ALTERNATE PRECEDENCE formula A_not_alternate_precedence_B(A: activity, B: activity)	B_never_between_A(B,A)**;	
16. NEGATION ALTERNATE SUCCESSION A_not_alternate_succession_B(A: activity, B: activity)	(A_not_alternate_precedence_B(A,B) ∧ A_not_alternate_response_B(A,B));	
17.a. NEGATION CHAIN RESPONSE formula chain_A_response_notB(A: activity, B: activity)	□((activity == A -> _O(activity != B)));	
17.b. NEGATION CHAIN PRECEDENCE formula chain_notA_precedence_B(A: activity, B: activity)	□((_O(activity == B) -> activity != A));	
17.c. NEGATION CHAIN SUCCESSION formula chain_A_notsuccession_B(A: activity, B: activity)	(chain_A_response_notB(A,B) ∧ chain_notA_precedence_B(A,B));	
** subformula B_never_between_A(A: activity, B: activity)	□((activity == A -> _O((<>(activity == A) -> (activity != B _U activity == A)))));	

Figure F-3: *Negation Relations Formulas and their Notation, [van der Aalst and Pesic, 2006]*

APPENDIX G

SCIFF AND CLP

We have presented in Chapter 6 Section 6.4 how MSMAS models can be verified at design time using $\mathcal{S}$CIFF framework, in this appendix we present the framework in more detail.

G.1 The $\mathcal{S}$CIFF Framework

$\mathcal{S}$CIFF is a logic programming framework originally proposed by Alberti et al. [2008]. It is based on Abductive Logic Programming (ALP) [Kakas et al., 1993]. An ALP is a triple $\langle P, A, IC\rangle$, where A is a set of predicates, named *abducibles*, P is a logic program that uses predicates in A but does not define them, and IC is a set of integrity constraints.

Reasoning in abductive logic programming is a goal-directed task (G, a goal), and amounts to finding an explanation set Δ of (ground) abducible predicates, such that: $P \cup \Delta \models G$ and $P \cup \Delta$ is consistent. The set IC of integrity constraints constrains the explanations Δ for the goal G, through the additional requirement $P \cup \Delta \models IC$.

$\mathcal{S}$CIFF leverages on ALP to constrain the dynamics of an event-based system, such as the interaction between multiple agents [Alberti et al., 2008] or the execution of a business process [Montali, 2010]. In particular, $\mathcal{S}$CIFF instantiates the ALP triple $\langle P, A, IC\rangle$ as follows:

- A is constituted by special predicates denoting expectations about (un)desired events;
- P is a knowledge base used to capture the static knowledge of the targeted system;
- IC is used to relate the occurrence of events to expected events, thus defining which are the events that are expected to occur when a certain trace of events is observed.

Events.

In $\mathcal{S}$CIFF there is a clear distinction between the description of an event and the occurrence of said event. In fact, an event is presented as a term, whereas an event that has happened is an atom $\mathbf{H}(Event, Time)$ where $Event$ is a $Term$ and $Time$ is an integer denoting the

Table G.1: *Syntax of events and expectations in $\mathcal{S}$CIFF*

$Event$	::=	$\mathbf{H}(GroundTerm[, Integer])$
$Expectation$	::=	$PosExp \mid NegExp$
$PosExp$	::=	$\mathbf{E}(Term[, Variable \mid Integer])$
$NegExp$	::=	$\mathbf{EN}(Term[, Variable \mid Integer])$
$Literal$	::=	[**not**]$Atom$
$AducibleLiteral$	::=	[**not**]$AducibleAtom$

time at which that event happened. Ground happened events are used to present a (partial) execution trace of the system, enumerating the relevant events and their timestamps, whereas happened events with variables are used to denote a class of matching ground happened events. For example, $\mathbf{H}(inform(john, mary, call\_code(123)), 5)$ denotes that $john$ informed $mary$ at time 5 that the call code has value 123. Whereas, $\mathbf{H}(inform(X, mary, call\_code(C)), T)$ models that some agent X informed $mary$ at a certain time T that the call code has value C.

As well as happened events, $\mathcal{S}$CIFF supports the modelling of (un)desired courses of interaction by introducing the notion of *expected* events, making it possible to explicitly describe what is expected (not) to happen. Expectations can be either positive $\mathbf{E}(Event, Time)$ or negative $\mathbf{EN}(Event, Time)$. The intuitive quantification for the variables possibly contained in the expectations is existential for positive expectations, and universal for negative expectations. For example, $\mathbf{E}(inform(X, mary, call\_code(C)), T)$ models that it is expected that someone informs $mary$ about the call code at some point in time, whereas $\mathbf{EN}(inform(X, mary, call\_code(C)), T)$ means that no agent can ever inform $mary$ about call codes. The full $\mathcal{S}$CIFF event syntax appears in [Alberti et al., 2008].

Table G.1 shows the full syntax of events in $\mathcal{S}$CIFF.

G.1.1 $\mathcal{S}$CIFF Integrity Constraints

In the $\mathcal{S}$CIFF framework, integrity constraints (ICs) are used to express behavioural rules interconnecting happened events with expectations, to present the expected and forbidden courses of interaction when a given pattern of happened events is found in the current system trace. Technically, they are (forward) implications of the form $\beta(\vec{X}) \rightarrow \gamma(\vec{X}, \vec{Y})$, where $\beta(\vec{X})$ is a conjunction of literals, i.e., of (partially grounded) happened/expected events and other predicates, and $\gamma(\vec{X}, \vec{Y})$ is a disjunction or conjunction of expectations and other predicates. Intuitively, variables $\vec{X}$ are universally quantified with the entire implication as scope, whereas variables $\vec{Y}$ are existentially or universally quantified depending on whether they appear inside positive or negative expectations (for a full account of quantification, see [Alberti et al., 2008]). Predicates are used to constrain further the matching events, and include Constraint

Table G.2: *Syntax of Integrity Constraints in $\mathcal{S}$CIFF*

ICs	::=	$[IC]^\star$
IC	::=	$Body \rightarrow Head$
$Body$	::=	$(Event \mid Expectation \mid AducibleLiteral)[\wedge BodyLiteral]^\star$
$BodyLiteral$	::=	$Event \mid ExtLiteral$
$Head$	::=	$HeadDisjunct[\vee HeadDisjunct]^\star \mid false$
$HeadDisjunct$	::=	$ExtLiteral[\wedge ExtLiteral]^\star$
$ExtLiteral$	::=	$Literal \mid AducibleLiteral \mid Expectation \mid Constraint$

logic programming (CLP)[1] constraints. When applied to time variables, CLP constraints are particularly useful for imposing metric temporal conditions on happened/expected events. For example, the integrity constraint:

$$\mathbf{H}(create\_call(X, C), T) \wedge friend(X, Y) \rightarrow \mathbf{E}(inform(X, Y, call\_code(C)), T_2) \wedge T_2 < T + 10$$

states that whenever agent X creates a call with code C, X is expected to inform each of her friends about the value of the call code within 10 time units. Once again, for the full $\mathcal{S}$CIFFsocial integrity constraint syntax, see [Alberti et al., 2008].

Table G.2 shows the full syntax of Social Integrity Constraints in $\mathcal{S}$CIFF.

G.1.2 $\mathcal{S}$CIFF Knowledge Base

$\mathcal{S}$CIFF ICs can capture the dynamic aspects of a system by interconnecting the observed and expected courses of interaction. However, they are not meant to present the static knowledge that might be needed to describe the system independently of its dynamics. The $\mathcal{S}$CIFF framework allows the definition of this type of knowledge inside a knowledge base (KB). The KB can be used to list facts known about the domain under study (such as the extension of the $friend$ predicate used in the aforementioned sample integrity constraint), or to encode complex derivation rules modeled as logic programming clauses. Such derivation rules could also employ happened and expected events to provide a-priori definitions for knowledge related to the system dynamics. The full syntax for $\mathcal{S}$CIFF knowledge base terms is given in [Alberti et al., 2008].

Table G.3 shows the full syntax of Social Integrity Constraints in $\mathcal{S}$CIFF.

G.1.3 Compliance in $\mathcal{S}$CIFF

We now describe the declarative semantics of $\mathcal{S}$CIFF, which builds upon the semantics of ALP and extends it so as to capture the meaning of expectations. In particular, $\mathcal{S}$CIFF declaratively captures the notion of *compliance* of a system execution trace with the modelled specification.

[1] A Constraint Logic Program is a logic program that contains constraints in the body of clauses.

Table G.3: *Syntax of the Knowledge Base in $\mathcal{S}$CIFF*

KB_{DSF}	::=	$[Clause]^\star$
$Clause$	::=	$Head \leftarrow Body$
$Head$	::=	$Atom$
$Body$	::=	$ExtLiteral[\wedge ExtLiteral]^\star \mid true$
$ExtLiteral$	::=	$Literal \mid AducibleLiteral \mid ExpLiteral \mid Constraint$

This is done by considering positive and negative expectations as abducible predicates, and by introducing the notion of *fulfillment*. Starting from the knowledge base and the set of happened events contained in the analyzed trace of the system (which extends the knowledge base with information about the dynamics), expectations are hypothesized consistently with the ICs and with an expectation-consistency rule stating that no event can be expected to happen and not to happen at the same time. A positive (respectively negative) expectation is then judged as fulfilled (respectively violated) if there exists a corresponding matching happened event in the trace. This can be considered as a sort of *hypothesis confirmation* step, where the hypothesized courses of execution match with an actual behaviour.

To deal with the possibility that only a partial trace of the system is known, $\mathcal{S}$CIFF can maintain the generated expectations *pending*, awaiting further happened events to judge their fulfillment. A special case is when it is known that no further happened event will ever occur. In this case, all pending positive expectations are considered violated, whereas all pending negative expectations are considered fullfilled: no matching event will ever be found in the future.

This declarative notion of compliance has an operational counterpart in the $\mathcal{S}$CIFF proof procedure, which concretely realizes an inference mechanism to 1. dynamically acquire happened events reporting about the evolution of the system dynamics, 2. use the modeled knowledge base and integrity constraints so as to generate expectations about the courses of execution, and 3. match expectations with happened events, deciding their fulfillment. Execution traces which fulfill all the generated expectations are then deemed as *compliant* with the specification.

An extension of the $\mathcal{S}$CIFF proof procedure, called g-$\mathcal{S}$CIFF, can be used to prove properties of the model at design time, i.e., without having an explicit trace of the system [Montali et al., 2010b]. Given a property, g-$\mathcal{S}$CIFF tries to generate a (partially specified) trace showing that the property can be satisfied while respecting all the modeled ICs. Intuitively, this is done by transforming every pending positive expectation into a corresponding happened event, checking that no negative expectation becomes violated.

Finally, we observe that while termination of the proof procedures cannot be guaranteed in general, all the techniques developed to check termination of (abductive) logic programs can be seamlessly applied to $\mathcal{S}$CIFF (see, e.g., [Montali, 2010, Montali et al., 2010b] for a discussion on termination conditions when reasoning on extended DECLARE.

APPENDIX H

LIST OF MSMAS NORMS FORMALISM IN CLIMB

Chapter 6 have the representation of only one MSMAS norm group (goal/gole relations) in CLIMB. In this appendix we present the mapping of the remaining three norm groups. For the full list of MSMAS events please refer to Chapter 6.

H.1 Role/Role Relation Formalisation in CLIMB

Given a MSMAS model, each Role/Role relation present in that model is captured as a corresponding fact of the type:

$role\_role\_relation(A, B, Type)$

Where A and B are institutional roles and $Type$ is the type of role/role relation. For example, $role\_role\_relation(IR\_System\_Admin, IR\_Monitoring\_Service, childParent\_roles)$, expresses that a child/parent roles relation holds between IR_System_Admin and $IR\_Monitoring\_Service$ roles. All these facts are grouped together inside an "Institutional Roles Norms" knowledge base $\mathcal{KB}_{irn}$.

The role/role relations described in Chapter 4 in Figure 4-12 are then formalised by means of $\mathcal{IC}$s that follow the ConDec^{++} to CLIMB translation presented in [Montali, 2010]. Such constraints are grouped together into an integrity constraint set $\mathcal{IC}_{irn}$.

Sequential Roles (SR):

$$role\_role\_relation(A, B, sequential\_roles)$$
$$\wedge\, \mathbf{H}(play\_role(A, SP), T_B)$$
$$\rightarrow \mathbf{E}(play\_role(B, SP), T_A) \wedge T_A < T_B.$$

Notice that the constraint is instantiated for every role/role relation of type $sequential\_roles$ contained into $\mathcal{KB}_{msmas}$.

Sequential Roles (SR) with Time Constraint (n, m):

$$role\_role\_relation(A, B, sequential\_roles, (n, m))$$
$$\wedge \mathbf{H}(play\_role(A, SP), T_B)$$
$$\rightarrow \mathbf{E}(play\_role(B, SP), T_A) \wedge T_B > T_A + n \wedge T_B <= T_A + m.$$

Joint Roles (JR):

$$role\_role\_relation(A, B, joint\_roles)$$
$$\wedge \mathbf{H}(play\_role(A, SP), T_A)$$
$$\rightarrow \mathbf{E}(play\_role(B, SP), T_B) \wedge T_B > T_A.$$
$$role\_role\_relation(A, B, joint\_roles)$$
$$\wedge \mathbf{H}(play\_role(B, SP), T_B)$$
$$\rightarrow \mathbf{E}(play\_role(A, SP), T_A) \wedge T_A < T_B.$$

Joint Roles (JR) with Time Constraint (n, m):

$$role\_role\_relation(A, B, joint\_roles, (n, m))$$
$$\wedge \mathbf{H}(play\_role(A, SP), T_A)$$
$$\rightarrow \mathbf{E}(play\_role(B, SP), T_B) \wedge T_B > T_A + n \wedge T_B <= T_A + m.$$
$$role\_role\_relation(A, B, joint\_roles, (n, m))$$
$$\wedge \mathbf{H}(play\_role(B, SP), T_B)$$
$$\rightarrow \mathbf{E}(play\_role(A, SP), T_A) \wedge T_B > T_A + n \wedge T_B <= T_A + m.$$

Notice that the constraint is instantiated for every goal/goal relation of type $sequential\_roles$ contained into $\mathcal{KB}_{msmas}$.

Coupled Roles (CR):

$$role\_role\_relation(A, B, coupled\_roles)$$
$$\wedge \mathbf{H}(play\_role(A, SP), T_A)$$
$$\rightarrow \mathbf{E}(play\_role(B, SP), T_B)$$
$$\wedge \mathbf{EN}(drop\_role(A, SP), T_C) \wedge T_C < T_B.$$
$$goal\_goal\_relation(A, B, coupled\_roles)$$
$$\wedge \mathbf{H}(play\_role(B, SP), T_B)$$
$$\rightarrow \mathbf{E}(play\_role(A, SP), T_A)$$
$$\wedge \mathbf{EN}(drop\_role(B, SP), T_C) \wedge T_C < T_A.$$

Notice the use of $drop\_role$ event to specify that both roles have to be played at the same time for some time.

Disjoint Roles (DR):

$$role\_role\_relation(A, B, disjoint\_roles)$$
$$\wedge \mathbf{H}(play\_role(A, SP), T_A)$$
$$\rightarrow \mathbf{EN}(play\_role(B, SP), T_B).$$
$$role\_role\_relation(A, B, disjoint\_roles)$$
$$\wedge \mathbf{H}(play\_role(B, SP), T_B)$$
$$\rightarrow \mathbf{EN}(play\_role(A, SP), T_A).$$

Notice that the constraint is instantiated for every goal/goal relation of type $disjoint\_roles$ contained into $\mathcal{KB}_{msmas}$. The next $\mathcal{IC}$ is to demonstrate how to utilise the child/parent relation between institutional roles using the defined system goal events $delegate\_goal$ and $satisfyDelegated\_goal$

ChildParent Roles (CPR):

$$role\_role\_relation(A, B, childParent\_roles)$$
$$\wedge \mathbf{H}(delegate\_goal((A, IR_A), (B, IR_B), SG, T_A)).$$
$$\rightarrow \mathbf{E}(satisfyDelegated\_goal((B, IR_B), A, IR_A), SG, T_B)) \wedge T_B > T_A.$$

Notice that the constraint is instantiated for every goal/goal relation of type $childParent\_goals$ contained into $\mathcal{KB}_{msmas}$.

H.2 Message/Message Relation Formalisation in CLIMB

Given a MSMAS Communication Protocol model, each Message/Message relation present in the model is captured as a corresponding fact of the type:

$message\_message\_relation(A, B, Type)$

Where A and B are communication messages and $Type$ is the type of message/message relation. For example,
$message\_message\_relation(MSG\_Agree, MSG\_SendCustomFile, sequential\_messages)$, expresses that a precedence/sequential messages relation holds between MSG_Agree and $MSG\_SendCustomFile$ messages. All these facts are grouped together inside a "Communication Protocols Norms" knowledge base $\mathcal{KB}_{cpn}$.

The message/message relations described in Chapter 4 in Figure 4-23 are then formalised by means of $\mathcal{IC}$s that follow the ConDec^{++} to CLIMB translation presented in [Montali, 2010]. Such constraints are grouped together into an integrity constraint set $\mathcal{IC}_{cpn}$.

Sequential Messages (SM):

$$message\_message\_relation(A, B, sequential\_messages)$$
$$\wedge\, \mathbf{H}(send\_message(MT_B, B, (SID_B, SIRB), (RID_A, RIR_A), MSG_B, CP), T_B).$$
$$\rightarrow \mathbf{E}(send\_message(MT_A, A, (SID_C, SIRC), (RID_D, RIR_D), MSG_A, CP), T_A)$$
$$\wedge\, T_A < T_B.$$

Notice that the $\mathcal{IC}$ allows for setting relations between messages that are exchanged between the same interacting parties and completely different interacting parties. The specification of the communication protocol about the acceptable institutional roles associated with each message can be specified using the $\mathcal{KB}_{cpn}$ being a static knowledge. The constraint is instantiated for every message/message relation of type $sequential\_messages$ contained into $\mathcal{KB}_{cpn}$.

Sequential Messages (SM) with Time Constraint (n, m):

$$message\_message\_relation(A, B, sequential\_messages)$$
$$\wedge\, \mathbf{H}(send\_message(MT_B, B, (SID_B, SIRB), (RID_A, RIR_A), MSG_B, CP), T_B).$$
$$\rightarrow \mathbf{E}(send\_message(MT_A, A, (SID_C, SIRC), (RID_D, RIR_D), MSG_A, CP), T_A)$$
$$\wedge\, T_B > T_A + n \wedge T_B <= T_A + m.$$

Joint Messages (JM):

$$message\_message\_relation(A, C, joint\_messages)$$
$$\wedge\, \mathbf{H}(send\_message(MT_A, A, (SID_A, SIRA), (RID_B, RIR_B), MSG_A, CP), T_A)$$
$$\rightarrow \mathbf{E}(send\_message(MT_C, C, (SID_C, SIRC), (RID_D, RIR_D), MSG_C, CP), T_C)$$
$$\wedge\, T_C > T_A.$$
$$message\_message\_relation(A, B, joint\_message)$$
$$\wedge\, \mathbf{H}(send\_message(MT_C, C, (SID_C, SIRC), (RID_D, RIR_D), MSG_C, CP), T_C)$$
$$\rightarrow \mathbf{E}(send\_message(MT_A, A, (SID_A, SIRA), (RID_B, RIR_B), MSG_A, CP), T_A)$$
$$\wedge\, T_A < T_C.$$

Joint Messages (JM) with Time Constraint (n, m):

$$message\_message\_relation(A, C, joint\_messages)$$
$$\wedge\, \mathbf{H}(send\_message(MT_A, A, (SID_A, SIRA), (RID_B, RIR_B), MSG_A, CP), T_A)$$
$$\rightarrow \mathbf{E}(send\_message(MT_C, C, (SID_C, SIRC), (RID_D, RIR_D), MSG_C, CP), T_C)$$
$$\wedge\, T_C > T_A + n \wedge T_C <= T_A + m.$$
$$message\_message\_relation(A, B, joint\_message)$$
$$\wedge\, \mathbf{H}(send\_message(MT_C, C, (SID_C, SIRC), (RID_D, RIR_D), MSG_C, CP), T_C)$$
$$\rightarrow \mathbf{E}(send\_message(MT_A, A, (SID_A, SIRA), (RID_B, RIR_B), MSG_A, CP), T_A)$$
$$\wedge\, T_C > T_A + n \wedge T_C <= T_A + m.$$

Notice that the constraint is instantiated for every message/message relation of type $sequential\_messages$ contained into $\mathcal{KB}_{cpn}$.

Coupled Messages (CM):

$$message\_message\_relation(A, C, coupled\_messages)$$
$$\wedge\, \mathbf{H}(send\_message(MT_A, A, (SID_A, SIRA), (RID_B, RIR_B), MSG_A, CP), T_A)$$
$$\rightarrow \mathbf{E}(send\_message(MT_C, C, (SID_C, SIRC), (RID_D, RIR_D), MSG_C, CP), T_C).$$
$$message\_message\_relation(A, C, coupled\_messages)$$
$$\wedge\, \mathbf{H}(send\_message(MT_C, C, (SID_C, SIRC), (RID_D, RIR_D), MSG_C, CP), T_C)$$
$$\rightarrow \mathbf{E}(send\_message(MT_A, A, (SID_A, SIRA), (RID_B, RIR_B), MSG_A, CP), T_A).$$

Notice that the constraint is instantiated for every goal/goal relation of type $coupled\_messages$ contained into $\mathcal{KB}_{cpn}$.

Disjoint Messages (DM):

$$goal\_goal\_relation(A, C, disjoint\_messages)$$
$$\wedge\, \mathbf{H}(send\_message(MT_A, A, (SID_A, SIRA), (RID_B, RIR_B), MSG_A, CP), T_A)$$
$$\rightarrow \mathbf{EN}(send\_message(MT_C, C, (SID_C, SIRC), (RID_D, RIR_D), MSG_C, CP), T_C).$$
$$message\_message\_relation(A, C, coupled\_messages)$$
$$\wedge\, \mathbf{H}(send\_message(MT_C, C, (SID_C, SIRC), (RID_D, RIR_D), MSG_C, CP), T_C)$$
$$\rightarrow \mathbf{EN}(send\_message(MT_A, A, (SID_A, SIRA), (RID_B, RIR_B), MSG_A, CP), T_A).$$

Notice that the constraint is instantiated for every goal/goal relation of type $disjoint\_goals$ contained into $\mathcal{KB}_{cpn}$.

As explained earlier some communication protocols require relationship branching, which is expressed as a disjunctions expected messages. For example below is the $\mathcal{IC}$ to express the branching sequntial_goals relation between MSG_RequestFile and (MSG_Agree, MSG_Refuse, and MSG_Error) messages in CCP_TranslateFile communication protocol shown in Figure H-1:

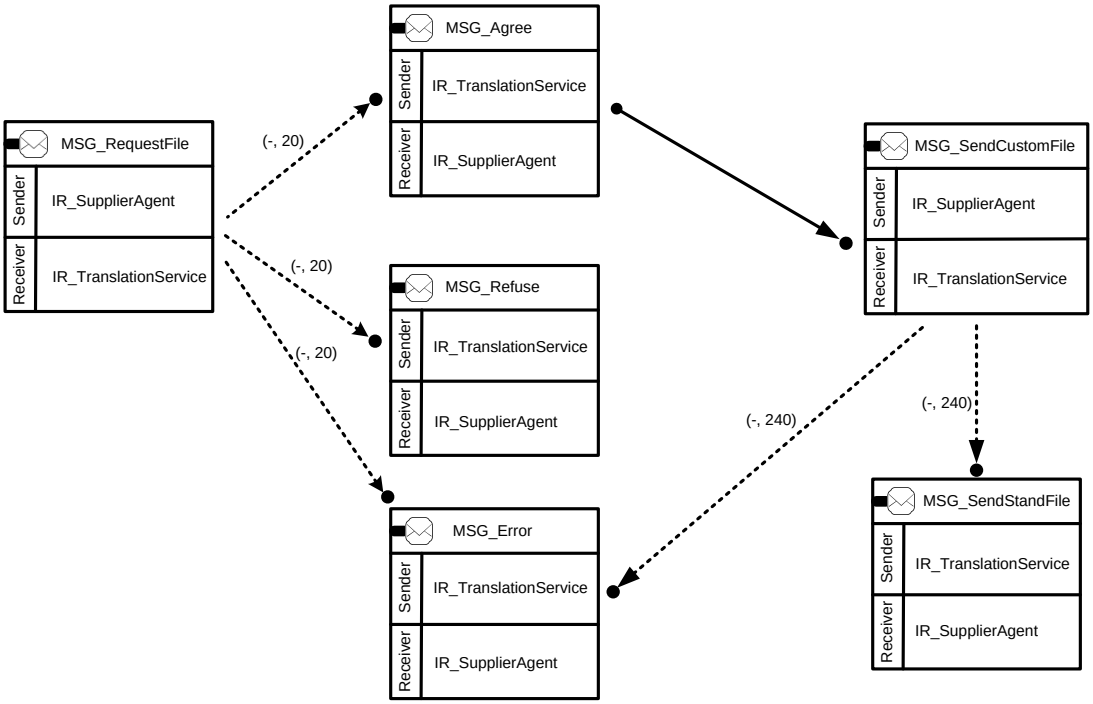


Figure H-1: *Custom Communication Protocol Model (CCP_TranslateFile)*

Sequential Messages (SM) with Time Constraint (n, m):

$$
\begin{aligned}
&message\_message\_relation(MSG\_RequestFile, (MSG\_Agree, MSG\_Refuse, MSG\_Error),\\
&\quad sequential\_messages)\\
&\wedge \mathbf{H}(send\_message(informMessage, MSG\_Agree, (SP_{10}, IR\_TranslationService),\\
&\quad (SP_{150}, IR\_SupplierAgent), 502Agree, CCP\_TranslateFile), T_B)\\
&\vee \mathbf{H}(send\_message(informMessage, MSG\_Refuse, (SP_{10}, IR\_TranslationService),\\
&\quad (SP_{150}, IR\_SupplierAgent), 503Refuse, CCP\_TranslateFile), T_B)\\
&\vee \mathbf{H}(send\_message(informMessage, MSG\_Error, (SP_{10}, IR\_TranslationService),\\
&\quad (SP_{150}, IR\_SupplierAgent), 504Error), CCP\_TranslateFile), T_B).\\
&\quad \rightarrow \mathbf{E}(send\_message(RequestMessage, MSG\_RequestFile,\\
&\quad (SP_{150}, IR\_SupplierAgent), (SP_{10}, IR\_TranslationService), 501Request, CCP\_TranslateFile), T_A)\\
&\quad \wedge T_B > T_A + 0 \wedge T_B <= T_A + 20.
\end{aligned}
$$

Notice that the constraint is instantiated for every goal/goal relation of type $childParent\_goals$ contained into $\mathcal{KB}_{cpn}$.

H.3 Activity/Activity Relation Formalisation in CLIMB

MSMAS allows the use of all set of activity to activity relations as defined in ConDec^{++}. There are twenty six different constraint types defined by ConDec, however Chesani et al. [2009b]

showed that the basic existence, relation and negation constraints can be expressed in terms of only eight core constraints.

- the two basic cardinality constraints (*existence* and *absence*).
- the three fundamental positive temporal orderings (*responded existence* for any ordering, response for the after ordering, *precedence* for the before ordering).
- the *negation response* constraint.
- the *positive/negative interposition*[1] patterns.

The definition of $entry$ activity means that the triggering event $start\_activity$ is also firing the event $achieve\_goal$ of the basic system goal associated with this Basic Business Process. Likewise the definition of $exit$ activity means that the triggering event $end\_activity$ is also firing the event $satisfy\_goal$ of the basic system goal associated with this Basic Business Process. For a comprehensive treatment of such constraints in $\mathcal{S}$CIFF, see Montali [2010].

[1] Chesani et al. [2009b] presented "the concept of positive and negative interposition, where *interposition*(a,b,c) states that between any execution of activity a and a future execution of activity c, b must be performed at least once. While *negation interposition*(a,b,c) expresses the opposite constraint, specifying that the execution of a and a following c cannot be interleaved by b. X is sometimes used to represent an arbitrary activity."

APPENDIX

FULL LIST OF MSMAS NORMS FORMALISM AS EC AXIOMS

We have presented in Chapter 6 two system norms types as an example of mapping MSMAS norms into EC theories. In this appendix we present the the full list of two system norm groups and their mappings. The first group is Goal/Goal Relations which shares the same life cycle of Activity/Activity Relations, which the second group is Role/Role Relations which shares the same life cycle of Message/Message Relations.

I.1 MSMAS Triggering Events

The full list of triggering events is:

System Goals:	$(achieveGoal - satisfyGoal - dropGoal)$
Business Activities:	$(startActivity - endActivity - cancelActivity)$
Institutional Roles:	$(playRole - dropRole)$
Communication Messages:	$(sendMessage - cancelMessage)$

I.2 Goal/Goal Relation Formalisation in EC

Sequential Goals System Norm:

A Sequential Goals relation between two system goals (source: $goal_A$ - target: $goal_B$) states that $goal_B$ must be achieved after successfully achieving $goal_A$. This means the initial state of such a SN should be set to $satisfied$ (*Axiom 1*) then within an execution trace for each instance that completes successfully $goal_A$ the fluent representing this system norm instance should move to $pending$ state (*Axiom 2*), waiting for a complete event that successfully completes $goal_B$ to move to state $satisfied$ (*Axiom 3*) or waiting for the case to reach an end, all pending instances are then declared as $satisfied$ (*Axiom 5*). Only one case leads to a violation of the SN: (i) if it is in its initial state $satisfied$ then an event of successful completion of

$goal_B$ occurs (*Axiom 4*) or When the case reaches an end, all pending instances are declared as satisfied. This is a generic axiom that applies for all pending SNs, as it attests

The following axioms model these different cases:

Axiom 1 (Sequential Goals Creation). *Each sequential goals system norm is associated to a unique instance, created and put in the satisfied state when the case is started.*:

$$init\_state(i(id, start, 0), sat).$$

Axiom 2 (Sequential Goals Pending). *A satisfied sequential goals SN instance becomes* $pending$ *when its source system goal is completed.*:

$$trans(complete(-, A), T, i(id, A, T_i), sat, pend) \leftarrow$$
$$cur\_state(i(id, A, T_i), sat, T) \wedge T \geq T_i.$$

Axiom 3 (Sequential Goals Fulfilment). *A pending joint goals SN instance becomes* $satisfied$ *when its target system goal is completed.*:

$$trans(complete(-, B), T, i(id, A, T_i), pend, sat) \leftarrow$$
$$cur\_state(i(id, A, T_i), pend, T) \wedge T \geq T_i.$$

Axiom 4 (Sequential Goals Violation). *A satisfied joint goals SN instance becomes* $violated$ *when its target system goal is completed. Being in* $satisfied$ *state implicitly indicate the the source system goal is not completed yet.*:

$$trans(complete(-, B), T, i(id, A, T_i), sat, viol) \leftarrow$$
$$cur\_state(i(id, A, T_i), sat, T) \wedge T \geq T_i.$$

Axiom 5 (Semantics of Pending-based Fulfilment). *When the case reaches an end, all pending instances of Sequential Goals are declared as Satisfied.*:

$$trans(ev(-, d, case\_complete), T, i(-, -, -), pend, sat) \leftarrow$$
$$cur\_state(i(id, A, T_i), pend, T) \wedge T \geq T_i.$$

Joint Goals System Norm:

A Joint Goals relation between two system goals (source: $goal_A$ - target: $goal_B$) states that $goal_B$ must be achieved after successfully achieving $goal_A$, and $goal_A$ must be successfully achieved before achieving $goal_B$. This means the initial state of such a SN should be set to $satisfied$ (*Axiom 6*) then within an execution trace for each instance that completes successfully $goal_A$ the fluent representing this system norm instance should move to $pending$ state (*Axiom 7*), waiting for a complete event that successfully completes $goal_B$ to move to state $satisfied$ (*Axiom 8*). Two cases leads to a violation of the SN: (i) if it is in its initial state $satisfied$ then an event of successful completion of $goal_B$ occurs (*Axiom 9*) or (ii) when it is pending, and a case complete event is received that announces that no further events to happen

(*Axiom 10*). The following axioms model these different cases:

Axiom 6 (Joint Goals Creation). *Each joint goals system norm is associated to a unique instance, created and put in the satisfied state when the case is started.*:

$$init\_state(i(id, start, 0), sat).$$

Axiom 7 (Joint Goals Pending). *A satisfied joint goals SN instance becomes* $pending$ *when its source system goal is completed.*:

$$trans(complete(-, A), T, i(id, A, T_i), sat, pend) \leftarrow$$
$$cur\_state(i(id, A, T_i), sat, T) \wedge T \geq T_i.$$

Axiom 8 (Joint Goals Fulfilment). *A pending joint goals SN instance becomes* $satisfied$ *when its target system goal is completed.*:

$$trans(complete(-, B), T, i(id, A, T_i), pend, sat) \leftarrow$$
$$cur\_state(i(id, A, T_i), pend, T) \wedge T \geq T_i.$$

Axiom 9 (Joint Goals Violation). *A satisfied joint goals SN instance becomes* $violated$ *when its target system goal is completed. Being in* $satisfied$ *state implicitly indicate the the source system goal is not completed yet.*:

$$trans(complete(-, B), T, i(id, A, T_i), sat, viol) \leftarrow$$
$$cur\_state(i(id, A, T_i), sat, T) \wedge T \geq T_i.$$

Axiom 10 (Semantics of Pending-based Violation). *When the case reaches an end, all pending instances are declared as violated. This is a generic axiom that applies for all pending SNs, as it attests that the case has reached it end and as a result no further events is expected to occur to move the instance from the pending to the satisfied state.*:

$$trans(ev(-, d, case\_complete), T, i(-, -, -), pend, viol).$$

Coupled Goals System Norm:

A Coupled Goals relation between two system goals (source: $goal_A$ - target: $goal_B$) states that both goals have to be achieved or none. This means the initial state of such a SN should be set to $satisfied$ (*Axiom 11*) then within an execution trace for each instance that completes successfully $goal_A$ or $goal_B$ the fluent representing this system norm instance should move to $pending$ state (*Axiom 12*), waiting for a complete event that successfully completes the other goal to move to state $satisfied$ (*Axiom 13*). The only case leads to a violation of the SN: (i) when it is pending, and a case complete event is received that announces that no further events to happen (*Axiom 14*). The following axioms model these different cases:

Axiom 11 (Coupled Goals Creation). *Each joint goals system norm is associated to a unique instance, created and put in the satisfied state when the case is started.*:

$$init\_state(i(id, start, 0), sat).$$

Axiom 12 (Coupled Goals Pending). *A satisfied joint goals SN instance becomes* $pending$ *when either the source system goal or the target system goal is completed.*:

$$trans(complete(-,-),T,i(id,-,T_i),sat,pend) \leftarrow$$
$$cur\_state(i(id,-,T_i),sat,T) \wedge T \geq T_i.$$

Axiom 13 (Coupled Goals Fulfilment). *A pending joint goals SN instance becomes* $satisfied$ *when its other system goal is completed.*:

$$trans(complete(-,B),T,i(id,A,T_i),pend,sat) \leftarrow$$
$$cur\_state(i(id,A,T_i),pend,T) \wedge T \geq T_i.$$

Axiom 14 (Coupled Goals Violation). *A pending joint goals SN instance becomes* $violated$ *when its target system goal is completed. Being in* $pending$ *state implicitly indicate the one of the system goals is not completed yet.*:

$$trans(ev(-,d,case\_complete),T,i(-,-,-),pend,viol).$$

Disjoint Goals System Norm:

A Disjoint Goals relation between two system goals (source: $goal_A$ - target: $goal_B$) states that only one of the goals can be achieved or none. This means the initial state of such a SN should be set to $satisfied$ (*Axiom 15*) then within an execution trace for each instance that completes successfully $goal_A$ or $goal_B$ the fluent representing this system norm instance should remain in $satisfied$ state (*Axiom 16*). The only case leads to a violation of the SN is when the other goal is achieved as well (*Axiom 17*).

Axiom 15 (Disjoint Goals Creation). *Each joint goals system norm is associated to a unique instance, created and put in the satisfied state when the case is started.*:

$$init\_state(i(id,start,0),sat).$$

Axiom 16 (Disjoint Goals fulfillment). *A satisfied disjoint goals SN instance remains* $satisfied$ *when either the source system goal or the target system goal is completed.*:

$$trans(complete(-,-),T,i(id,-,T_i),sat,sat) \leftarrow$$
$$cur\_state(i(id,-,T_i),sat,T) \wedge T \geq T_i.$$

Axiom 17 (Coupled Goals Violation). *A satisfied disjoint goals SN instance becomes* $violated$ *when its other system goal is completed.*:

$$trans(complete(-,B),T,i(id,A,T_i),sat,viol) \leftarrow$$
$$cur\_state(i(id,A,T_i),sat,T) \wedge T \geq T_i.$$

BIBLIOGRAPHY

Abdelaziz, T., Elammari, M., and Unland, R. (2007). A framework for the evaluation of agent-oriented methodologies. In *Innovations in Information Technology, 2007. IIT'07. 4th International Conference on*, pages 491–495. IEEE.

Al-Hashel, E., Balachandran, B. M., and Sharma, D. (2007). A comparison of three agent-oriented software development methodologies: Roadmap, prometheus, and mase. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 909–916. Springer.

Alberola, J. M., Such, J. M., Botti, V., Espinosa, A., and García-Fornes, A. (2013). A scalable multiagent platform for large systems. *Computer Science and Information Systems*, 10(1):51–77.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P. (2008). Verifiable agent interaction in abductive logic programming: The sciff framework. *ACM Trans. Comput. Logic*, 9(4):29:1–29:43.

Alberti, M., Gavanelli, M., Lamma, E., Chesani, F., Mello, P., and Torroni, P. (2006). Compliance verification of agent interaction: a logic-based software tool. *Applied Artificial Intelligence*, 20(2-4):133–157.

Aldewereld, H., Padget, J., Vasconcelos, W., Vázquez-Salceda, J., Sergeant, P., and Staikopoulos, A. (2010). Adaptable, Organization-Aware, Service-Oriented Computing. *IEEE Intelligent Systems*, 25(4):26–35.

Alonso, F., Frutos, S., Martínez, L., and Montes, C. (2005). Sonia: A methodology for natural agent development. In Gleizes, M.-P., Omicini, A., and Zambonelli, F., editors, *Engineering Societies in the Agents World V*, volume 3451 of *Lecture Notes in Computer Science*, pages 245–260. Springer Berlin Heidelberg.

Alvarez-Napagao, S., Aldewereld, H., Vázquez-Salceda, J., and Dignum, F. (2011). Normative monitoring: Semantics and implementation. In De Vos, M., Fornara, N., Pitt, J., and Vouros, G., editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, volume 6541 of *Lecture Notes in Computer Science*, pages 321–336. Springer Berlin Heidelberg.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., et al. (2003). Business process execution language for web services.

Andrighetto, G., Governatori, G., Noriega, P., and van der Torre, L. (2012). Normative Multi-Agent Systems (Dagstuhl Seminar 12111). *Dagstuhl Reports*, 2(3):23–49.

Anton, A. I. (1996). Goal-based requirements analysis. In *Requirements Engineering, 1996., Proceedings of the Second International Conference on*, pages 136–144. IEEE.

Antón, A. I., Dempster, J. H., and Siege, D. F. (2000). Deriving goals from a use case based requirements specification for an electronic commerce system. In *Proc. REFSQ*, pages 5–6.

Antoniou, G. (2004). *A semantic web primer*. the MIT Press.

Argente, E., Julian, V., and Botti, V. (2006). Multi-agent system development based on organizations. *Electronic Notes in Theoretical Computer Science*, 150(3):55 – 71.

Armstrong, J. and Helen, T. (2003). Making reliable distributed systems in the presence of software errors.

Atkinson, C. and Kühne, T. (2003). Model-Driven development: A metamodeling foundation. *IEEE Software*, 20(5):36–41.

Austen, J. L. (1962). How to do things with words. *Cambridge, Mass: Harvard UP*.

Baldoni, M., Baroglio, C., Marengo, E., and Patti, V. (2013). Constitutive and regulative specifications of commitment protocols: A decoupled approach. *ACM Trans. Intell. Syst. Technol.*, 4(2):22:1–22:25.

Balke, T., da Costa Pereira, C., Dignum, F., Lorini, E., Rotolo, A., Vasconcelos, W., and Villata, S. (2013). Norms in mas: Definitions and related concepts. In *Normative Multi-Agent Systems*, pages 1–31.

Balke, T., De Vos, M., and Padget, J. (2012). Normative run-time reasoning for institutionally-situated bdi agents. In Cranefield, S., van Riemsdijk, M., Vázquez-Salceda, J., and Noriega, P., editors, *Coordination, Organizations, Institutions, and Norms in Agent System VII*, volume 7254 of *Lecture Notes in Computer Science*, pages 129–148. Springer Berlin Heidelberg.

Bauer, B. and Müller, J. P. (2004). Methodologies and modelling languages. In Luck, M., Ashri, R., and d'Inverno, M., editors, *Agent-Based Software Development*, chapter 4, pages 77–131. Artech House.

Bauer, B., Müller, J. P., and Odell, J. (2001). Agent uml: A formalism for specifying multiagent software systems. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 91–103. Springer Berlin Heidelberg.

Bernon, C., Cossentino, M., Gleizes, M.-P., Turci, P., and Zambonelli, F. (2005). A study of some multi-agent meta-models. In Odell, J., Giorgini, P., and Müller, J. P., editors, *Agent-Oriented Software Engineering V*, volume 3382 of *Lecture Notes in Computer Science*, pages 62–77. Springer Berlin Heidelberg.

Beydoun, G., Low, G., and Bogg, P. (2012). Suitability assessment framework of agent-based software architectures. *Information and Software Technology*.

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J., Pavon, J., and Gonzalez-Perez, C. (2009). Faml: A generic metamodel for mas development. *Software Engineering, IEEE Transactions on*, 35(6):841–863.

Boella, G. and der Torre, L. (2006). Constitutive norms in the design of normative multiagent systems. In Toni, F. and Torroni, P., editors, *Computational Logic in Multi-Agent Systems*, volume 3900 of *Lecture Notes in Computer Science*, pages 303–319. Springer Berlin Heidelberg.

Boella, G., Pigozzi, G., and van der Torre, L. (2009). Five guidelines for normative multiagent systems. In *Proceedings of JURIX 2009 - The 22nd International Conference on Legal Knowledge and Information Systems*, Frontiers in Artificial Intelligence and Applications, Rotterdam, The Netherlands. IOS Press.

Boella, G., Van Der Torre, L., and Verhagen, H. (2006). Introduction to normative multiagent systems. *Computational & Mathematical Organization Theory*, 12(2-3):71–79.

Boella, G., Van Der Torre, L., and Verhagen, H. (2008). Introduction to the special issue on normative multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 17(1):1–10.

Boella, G. and van der Torre, L. W. (2004). Regulative and constitutive norms in normative multiagent systems. *Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR2004)*, 4:255–265.

Bordini, R. H., Dastani, M., and Winikoff, M. (2007). Current issues in multi-agent systems development. In *Proceedings of the 7th International Conference on Engineering Societies in the Agents World VII*, ESAW'06, pages 38–61, Berlin, Heidelberg. Springer-Verlag.

Bratman, M. (1987). *Intention, plans, and practical reason*. Harvard University Press.

Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., and Treur, J. (1997). Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6(1):67–94.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. (2004). TROPOS: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

Broersen, J., Cranefield, S., Elrakaiby, Y., Gabbay, D., Grossi, D., Lorini, E., Parent, X., van der Torre, L. W., Tummolini, L., Turrini, P., et al. (2013). Normative reasoning and consequence. *Normative Multi-Agent Systems*, page 33.

Bryl, V. (2009). *Supporting the design of socio-technical systems by exploring and evaluating design alternatives*. PhD thesis, University of Trento.

Bryl, V., Mello, P., Montali, M., Torroni, P., and Zannone, N. (2008a). $\mathcal{B}$-tropos. In *Computational Logic in Multi-Agent Systems*, pages 157–176. Springer.

Bryl, V., Mello, P., Montali, M., Torroni, P., and Zannone, N. (2008b). *$\mathcal{B}$-Tropos: Agent-oriented requirements engineering meets computational logic for declarative business process modeling and verification*, volume 5056 of *Lecture Notes in Computer Science*, pages 157–176. Springer Verlag.

Burrafato, P. and Cossentino, M. (2002). Designing a multi-agent solution for a bookstore with the passi methodology. In *In Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002*, pages 27–28.

Busetta, P., Ronnquist, R., Hodgson, A., and Lucas, A. (1999). Jack intelligent agents - components for intelligent agents in java.

Bush, G., Cranefield, S., and Purvis, M. (2001). The styx agent methodology.

Bussmann, S. K. (2003). *An Agent-Oriented Design Methodology for Production Control*. PhD thesis, University of Southampton.

Cardoso, H. L. and Oliveira, E. (2007). Institutional reality and norms: Specifying and monitoring agent organisations. *International Journal of Cooperative Information Systems*, 16(01):67–95.

Castelfranchi, C. (1995). Commitments: From individual intentions to groups and organizations. In *The 1st International Conference on Multiagent Systems (ICMAS)*, volume 95, pages 41–48.

Cernuzzi, L., Rossi, G., and Plata, L. (2002). On the evaluation of agent oriented modeling methods. In *Proceedings of Agent Oriented Methodology Workshop, Seattle*, volume 29.

Chesani, F., Mello, P., Montali, M., and Torroni, P. (2009a). Commitment tracking via the reactive event calculus. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 9, pages 91–96.

Chesani, F., Mello, P., Montali, M., and Torroni, P. (2009b). Verification of choreographies during execution using the reactive event calculus. In Bruni, R. and Wolf, K., editors, *Web Services and Formal Methods*, pages 55–72. Springer-Verlag, Berlin, Heidelberg.

Chesani, F., Mello, P., Montali, M., and Torroni, P. (2010). A logic-based, reactive calculus of events. *Fundamenta Informaticae*, 105(1):135–161.

Chopra, A. K. and Singh, M. P. (2004). Commitments for flexible business processes. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1362–1363. IEEE Computer Society.

Chopra, A. K. and Singh, M. P. (2009). An architecture for multiagent systems an approach based on commitments. In *Proceedings of the AAMAS Workshop on Programming Multiagent Systems (ProMAS 2009)*.

Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). *Model checking*. MIT press.

Cliffe, O., De Vos, M., and Padget, J. (2007a). Answer set programming for representing and reasoning about virtual institutions. In *Computational Logic in Multi-Agent Systems*, pages 60–79. Springer.

Cliffe, O., De Vos, M., and Padget, J. (2007b). Specifying and reasoning about multiple institutions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, pages 67–85. Springer.

Cockburn, A. (2000). Structuring Use cases with goals. *Journal of Object-oriented Programming*.

Colombetti, M. (2000). A commitment-based approach to agent speech acts and conversations. In *Proc. Workshop on Agent Languages and Communication Policies, 4th International Conference on Autonomous Agents (Agents 2000)*, pages 21–29.

Colombetti, M. and Verdicchio, M. (2002). An analysis of agent speech acts as institutional actions. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: part 3*, pages 1157–1164. ACM.

Cossentino, M., Gaud, N., Hilaire, V., Galland, S., and Koukam, A. (2010). Aspecs: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 20(2):260–304.

Cossentino, M. and Potts, C. (2002). Passi: A process for specifying and implementing multi-agent systems using uml.

Cysneiros, G. and Zisman, A. (2004). Refining the prometheus methodology with i. In *Proceedings of the Third International Workshop on Agent-Oriented Methodologies, held at OOPSLA*.

Dam, K. and Winikoff, M. (2004). Comparing agent-oriented methodologies. In Giorgini, P., Henderson-Sellers, B., and Winikoff, M., editors, *Agent-Oriented Information Systems*, volume 3030 of *Lecture Notes in Computer Science*, pages 78–93. Springer Berlin Heidelberg.

Dardenne, A., Van Lamsweerde, A., and Fickas, S. (1993). Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50.

Darimont, R. and Van Lamsweerde, A. (1996). Formal refinement patterns for goal-driven requirements elaboration. *ACM SIGSOFT Software Engineering Notes*, 21(6):179–190.

Dastani, M. (2004). Agentlink-iii technical forum group programming multiagent system promas.

Dastani, M. (2008). Programming multi-agent systems. In Fisher, M., Sadri, F., and Thielscher, M., editors, *CLIMA*, volume 5405 of *Lecture Notes in Computer Science*, pages 13–16. Springer.

Dastani, M., Hulstijn, J., Dignum, F., and Meyer, J.-J. C. (2004). Issues in multiagent system development. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 922–929. IEEE Computer Society.

De Vos, M., Padget, J., and Satoh, K. (2011). Legal modelling and reasoning using institutions. In Tojo, S., editor, *Proceedings of JURISIN 2010*, volume 6797 of *Lecture Notes in Computer Science*. Springer.

DeLoach, S. A. (2004). The mase methodology. In *Methodologies and software engineering for agent systems*, pages 107–125. Springer.

DeLoach, S. A., Wood, M. F., and Sparkman, C. H. (2001). Multiagent systems engineering. *Int. Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258.

Dignum, F. and Dignum, V. (2012). A formal semantics for agent (re) organization. In *Computational Logic in Multi-Agent Systems*, pages 61–76. Springer.

Dignum, M. (2003). *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, University of Utrech.

Dignum, V. and Padget, J. (2013). Multiagent organizations. In Weiss, G., editor, *Multiagent Systems*, pages 51–98. MIT Press, 2^{nd} edition. ISBN 978-0-262-01889-0. `http://mitpress.mit.edu/books/multiagent-systems-1` retrieved 20130309.

Dikenelli, O. et al. (2004). An infrastructure for the semantic integration of fipa compliant agent platforms. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3 (AAMAS'04)*, volume 3, pages 1316–1317. IEEE Computer Society.

Dinu, R., Stratulat, T., and Ferber, J. (2010). A formal approach to masq. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1443–1444. International Foundation for Autonomous Agents and Multiagent Systems.

d'Inverno, M. and Luck, M. (2004). *Understanding agent systems*. Springer Science & Business Media.

d'Inverno, M., Luck, M., Noriega, P., Rodríguez-Aguilar, J. A., and Sierra, C. (2012). Communicating open systems. *Artificial Intelligence*, 186:38–64.

Edmunson, S. A., Botterbusch, R. D., and Bigelow, T. A. (1992). Application of system modelling to the development of complex systems. In *Proceedings of the Digital Avionics Systems Conference*, pages 138 – 142. IEEE/AIAA 11th Vol, Issue, 5-8.

Elakehal, E., Montali, M., and Padget, J. (2014). Run-time verification of msmas norms using event calculus. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2014 IEEE Eighth International Conference on*, pages 110–115. IEEE.

Elakehal, E. E., Montali, M., and Padget, J. (2013). Verifying MSMAS model using $\mathcal{S}$ciff. In Klusch, M., Thimm, M., and Paprzycki, M., editors, *Multiagent System Technologies*, volume 8076 of *Lecture Notes in Computer Science*, pages 44–58. Springer Berlin Heidelberg.

Elakehal, E. E. and Padget, J. (2008). Pan-supplier stock control in a virtual warehouse (industry track). In *Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 11–18. ACM Press.

Elakehal, E. E. and Padget, J. (2012a). Market intelligence and price adaptation. In *Proceedings of the 14th Annual International Conference on Electronic Commerce*, ICEC '12, pages 9–16, New York, NY, USA. ACM.

Elakehal, E. E. and Padget, J. (2012b). Msmas: Modelling self-managing multi agent systems. *SCPE: Scalable Computing: Practice and Experience*, 13(2):121–137.

Elakehal, E. E. and Padget, J. A. (2011). A practical method for developing multi agent systems: Apmdmas. In Brazier, F. M. T., Nieuwenhuis, K., Pavlin, G., Warnier, M., and Badica, C., editors, *IDC*, volume 382 of *Studies in Computational Intelligence*, pages 11–20. Springer.

Endert, H., Hirsch, B., Küster, T., and Albayrak, S. (2007). Towards a mapping from bpmn to agents. In *Service-Oriented Computing: Agents, Semantics, and Engineering*, pages 92–106. Springer.

Esteva, M., De La Cruz, D., and Sierra, C. (2002). Islander: an electronic institutions editor. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, pages 1045–1052. ACM.

Esteva, M., Rodríguez-Aguilar, J. A., Arcos, J., Sierra, C., Noriega, P., Rosell, B., and de la Cruz, D. (2008). Electronic institutions development environment. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, pages 1657–1658.

Esteva, M., Rodríguez-Aguilar, J. A., Sierra, C., Garcia, P., and Arcos, J. L. (2001). On the formal specifications of electronic institutions. In *Agent Mediated Electronic Commerce, The European AgentLink Perspective.*, pages 126–147. Springer-Verlag.

Fasli, M. (2007). *Agent technology for e-commerce*. John Wiley & Sons Chichester.

Ferber, J. and Gutknecht, O. (1997). Aalaadin: A meta-model for the analysis and design of organizations in multi-agent systems.

Ferber, J., Gutknecht, O., and Michel, F. (2004). From agents to organizations: An organizational view of multi-agent systems. In *Agent-Oriented Software Engineering IV*, pages 214–230. Springer.

Ferber, J., Michel, F., and Baez, J. (2005). Agre: Integrating environments with organizations. In Weyns, D., Dyke Parunak, H., and Michel, F., editors, *Environments for Multi-Agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, pages 48–56. Springer Berlin Heidelberg.

Ferber, J., Stratulat, T., and Tranier, J. (2009). Towards an integral approach of organizations in multi-agent systems: the masq approach. *Multi-agent Systems: Semantics and Dynamics of Organizational Models in Virginia Dignum (Ed), IGI*.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). Kqml as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, pages 456–463. ACM.

FIPA, T. (2008). Fipa communicative act library specification. *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00037/SC00037J. html (30.6. 2004)*.

Fischer, K., Hahn, C., and Madrigal-Mora, C. (2007). Agent-oriented software engineering: a model-driven approach. *International Journal of Agent-Oriented Software Engineering*, 1(3):334–369.

Fornara, N., Cardoso, H., Noriega, P., Oliveira, E., Tampitsikas, C., and Schumacher, M. (2013a). Modelling agent institutions. In Ossowski, S., editor, *Agreement Technologies*, volume 8 of *Law, Governance and Technology Series*, pages 277–307. Springer Netherlands.

Fornara, N. and Colombetti, M. (2010). Representation and monitoring of commitments and norms using owl. *AI communications*, 23(4):341–356.

Fornara, N., Ježić, G., Kušek, M., Lovrek, I., Podobnik, V., and Tržec, K. (2013b). Semantics in multi-agent systems. In *Agreement Technologies*, pages 115–136. Springer.

Fornara, N., Viganò, F., Verdicchio, M., and Colombetti, M. (2008). Artificial institutions: a model of institutional reality for open multiagent systems. *Artificial Intelligence and Law*, 16(1):89–105.

Franklin, G. F., Powell, J. D., and Emami-Naeini, A. (2006). Feedback control of dynamics systems. *Pretince Hall Inc*.

Fuchs, N. and Robertson, D. (1996). Declarative specification. *The Knowledge Engineering Review*, 11(4):317–331.

Fuxman, A., Kazhamiakin, R., Pistore, M., and Roveri, M. (2003). Formal tropos: language and semantics. *University of Trento and IRST*, 55.

Fuxman, A., Pistore, M., Mylopoulos, J., and Traverso, P. (2001). Model checking early requirements specifications in tropos. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, pages 174–181.

García-Camino, A., Noriega, P., and Rodríguez-Aguilar, J. A. (2005). Implementing norms in electronic institutions. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 667–673. ACM.

Gardner, R. and Walker, J. (1994). *Rules, games, and common-pool resources*. University of Michigan Press.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., and Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54.

Garro, A. and Russo, W. (2010). easyabms: A domain-expert oriented methodology for agent-based modeling and simulation. *Simulation Modelling Practice and Theory*, 18(10):1453 – 1467. ¡ce:title¿Simulation-based Design and Evaluation of Multi-Agent Systems¡/ce:title¿.

Gascueña, J. M. and Fernández-Caballero, A. (2009). Prometheus and ingenias agent methodologies: A complementary approach. In Luck, M. and Gomez-Sanz, J., editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 131–144. Springer Berlin Heidelberg.

Genesereth, M., Fikes, R. E., Brachman, R., Gruber, T., Hayes, P., Letsinger, R., Lifschitz, V., Macgregor, R., Mccarthy, J., Norvig, P., and Patil, R. (1992). Knowledge interchange format version 3.0 reference manual.

Gervais, M.-P. (2003). Odac: An agent-oriented methodology based on odp. *Autonomous Agents and Multi-Agent Systems*, 7:199–228.

Gholami, M. F., Habibi, J., Shams, F., and Khoshnevis, S. (2010). Criteria-based evaluation framework for service-oriented methodologies. In *Computer Modelling and Simulation (UKSim), 2010 12th International Conference on*, pages 122–130. IEEE.

Ghorbani, A., Bots, P., Dignum, V., and Dijkema, G. (2013). Maia: a framework for developing agent-based social simulations. *Journal of Artificial Societies and Social Simulation*, 16(2):9.

Giannakopoulou, D. and Havelund, K. (2001). Automata-based verification of temporal properties on running programs. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 412–416. IEEE.

Gibbs, J. P. (1965). Norms: The problem of definition and classification. *American Journal of Sociology*, pages 586–594.

Giddens, A. (1984). *The constitution of society: introduction of the theory of structuration*. Univ of California Press.

Giorgini, P., Massacci, F., Mylopoulos, J., and Zannone, N. (2006). Requirements engineering for trust management: model, methodology, and reasoning. *International Journal of Information Security*, 5(4):257–274.

Glaser, N. (1997). The CoMoMAS methodology and environment for multi-agent system development. In Zhang, C. and Lukose, D., editors, *Multi-Agent Systems Methodologies and Applications*, volume 1286 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg.

Goknil, A., Kurtev, I., and Van Den Berg, K. (2014). Generation and validation of traces between requirements and architecture based on formal trace semantics. *Journal of Systems and Software*, 88:112–137.

Governatori, G. and Rotolo, A. (2010). Changing legal systems: Legal abrogations and annulments in defeasible logic. *Logic Journal of IGPL*, 18(1):157–194.

Grechanik, M., McKinley, K. S., and Perry, D. E. (2007). Recovering and using use-case-diagram-to-source-code traceability links. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 95–104. ACM.

Grossi, D., Dignum, F., Dastani, M., and Royakkers, L. (2005). Foundations of organizational structures in multiagent systems. In *Proceedings of AAMAS'05, Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 690–697. ACM Press.

Guo, L., Robertson, D., and Chen-Burger, Y.-H. (2008). Using multi-agent platform for pure decentralised business workflows. *Web Intelligence and Agent Systems*, 6(3):295–311.

Hahn, C., Madrigal-Mora, C., and Fischer, K. (2009). A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 18(2):239–266.

Hammer, M. and Champy, J. (1993). Reengineering the corporation: a manifesto for business evolution. *Nicholas Brealey, London*.

Hare, R. M. (1965). Critical studies. *The Philosophical Quarterly*, 15(59):172–175.

Hart, H. L. A. (2012). *The concept of law*. Oxford University Press.

Henderson-Sellers, B. (2013). Towards the consolidation of a diagramming suite for agent-oriented modelling languages. *ISRN Software Engineering*, 2013.

Henderson-Sellers, B. and Giorgini, P. (2005). *Agent-oriented methodologies*. IGI Global.

Highsmith, J. and Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*, 34(9):120–127.

Hill, D. R. C. (2002). Theory of modelling and simulation: Integrating discrete event and continuous complex dynamic systems: Second Edition by B. P. Zeigler, H. Praehofer, T. G. Kim, Academic Press, San Diego, CA, 2000. *International Journal of Robust and Nonlinear Control*, 12(1):91–92.

Hollingsworth, D. (1995). Workflow management coalition - the workflow reference model. Technical report, Workflow Management Coalition.

Hollingsworth, D. et al. (2004). The workflow reference model: 10 years on. In *Fujitsu Services, UK; Technical Committee Chair of WfMC*.

Hopton, L., Cliffe, O., Vos, M. D., and Padget, J. A. (2009). InstQL: A Query language for virtual institutions using answer set programming. In *Computational Logic in Multi-Agent Systems - 10th International Workshop, CLIMA X, Hamburg, Germany, September 9-10, 2009, Revised Selected and Invited Papers*, pages 102–121.

Horling, B. and Lesser, V. (2005). A Survey of Multi-Agent Organizational Paradigms. *The Knowledge Engineering Review*, 19(4):281–316.

Horn, P. (2001). Autonomic computing: IBM's perspective on the state of information technology.

Iglesias, C. A., Garijo, M., Centeno-González, J., and Velasco, J. R. (1998). Analysis and design of multiagent systems using MAS-Common KADS. In *Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*, ATAL '97, pages 313–327. Springer-Verlag.

Isern, D., Sánchez, D., and Moreno, A. (2011). Organizational structures supported by agent-oriented methodologies. *Journal of Systems and Software*, 84(2):169–184.

Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. (1992). *Object-oriented software engineering: a use case driven approach*. ACM press New York.

Jayaratna, N. (1994). *Understanding and evaluating methodologies: NIMSAD, a systematic framework*. McGraw-Hill, Inc.

Jennings, N. R. (1999). Agent-oriented software engineering. In *Multiple Approaches to Intelligent Systems*, pages 4–10. Springer.

Jennings, N. R., Norman, T. J., Faratin, P., O'Brien, P., and Odgers, B. (2000). Autonomous agents for business process management. *Applied Artificial Intelligence*, 14(2):145–189.

Jennings, N. R. and Sycara, K. (1998). A roadmap of agent research and development.

Jones, A. J. and Sergot, M. (1996a). A formal characterisation of institutionalised power. *Logic Journal of IGPL*, 4(3):427–443.

Jones, A. J. I. and Sergot, M. J. (1996b). A formal characterisation of institutionalised power. *Logic Journal of the IGPL*, 4(3):427–443.

Juan, T., Pearce, A., and Sterling, L. (2002). Roadmap: Extending the gaia methodology for complex open systems. In *AAMAS'02 Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 3–10. ACM Press.

Juran, J. M. and Gryna, F. M. (1970). *Quality planning and analysis*. McGraw-Hill New York.

Kakas, A. C., Kowalski, R. A., and Toni, F. (1993). Abductive Logic Programming.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Klügl, F. and Davidsson, P. (2013). AMASON: Abstract Meta-model for Agent-Based SimulatiON. In Klusch, M., Thimm, M., and Paprzycki, M., editors, *Multiagent System Technologies*, volume 8076 of *Lecture Notes in Computer Science*, pages 101–114. Springer Berlin Heidelberg.

Koestler, A. (1967). *The ghost in the machine*. Macmillan.

Kowalski, R. (1992). Database updates in the event calculus. *The Journal of Logic Programming*, 12(1):121–146.

Kowalski, R. and Sergot, M. (1989). A logic-based calculus of events. In *Foundations of knowledge base management*, pages 23–55. Springer.

Kowalski, R. A. and Sergot, M. J. (1986). A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95.

Kumar, A. and Goyal, V. (2012). Comparative analysis & evaluation of agent oriented methodologies using enhanced feature analysis framework. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(1).

Labrou, Y. and Finin, T. (1998). Semantics and conversations for an agent communication language. In Huhns, M. N. and Singh, M. P., editors, *Readings in Agents*, pages 235–242. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Laclavik, M., Balogh, Z., Gatial, E., and Hluchy, L. (2006). Agent architecture based on semantic knowledge model. In *The 5th Annual Conference. VSB-Technick*, pages 288–291. Citeseer.

Letier, E. and Van Lamsweerde, A. (2002). Agent-based tactics for goal-oriented requirements elaboration. In *Proceedings of the 24th International Conference on Software Engineering*, pages 83–93. ACM.

Li, T., Balke, T., De Vos, M., Satoh, K., and Padget, J. (2012). Conflict detection in composite institutions. In *International Workshop on Agent-based Modeling for Policy Engineering (AMPLE 2012)*, page 75.

Liu, L., Yu, E., and Mylopoulos, J. (2003). Security and privacy requirements analysis within a social setting. In *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 151–161. IEEE.

Lopez, y. F. L. and Luck, M. (2002). Towards a model of the dynamics of normative multi-agent systems. In *Proceedings of the International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02), volume 318 of Mitteilung*.

Luck, M., Griffiths, N., and d'Inverno, M. (1996). From agent theory to agent construction: A case study. In Müller, J. P., Wooldridge, M., and Jennings, N. R., editors, *Intelligent Agents III Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science*, pages 49–63. Springer.

Luck, M., McBurney, P., and Preist, C. (2003). Agent technology: Enabling next generation computing (a roadmap for agent based computing).

Luck, M., McBurney, P., Shehory, O., and Willmott, S. (2005). Agent technology: Computing as interaction (a roadmap for agent based computing).

Manna, Z. and Pnueli, A. (1992). *Temporal Logic*. Springer.

McNaughton, R. and Yamada, H. (1960). Regular expressions and state graphs for automata. *Electronic Computers, IRE Transactions on*, 1:39–47.

Mellor, S. (2004). *MDA Distilled: Principles of Model-driven Architecture*. Object Technology Series. Addison-Wesley.

Montali, M. (2010). *Specification and verification of declarative open interaction models: A logic-based approach*, volume 56. Springer Science & Business Media.

Montali, M., Chesani, F., Mello, P., and Maggi, F. M. (2013). Towards data-aware constraints in declare. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1391–1396, New York, NY, USA. ACM.

Montali, M., Maggi, F. M., Chesani, F., Mello, P., and van der Aalst, W. M. (2011). Monitoring business constraints with the event calculus. *ACM Transactions on Intelligent Systems and Technology*.

Montali, M., Pesic, M., Aalst, W. M. P. v. d., Chesani, F., Mello, P., and Storari, S. (2010a). Declarative specification and verification of service choreographiess. *ACM Trans. Web*, 4(1):3:1–3:62.

Montali, M., Torroni, P., Chesani, F., Mello, P., Alberti, M., and Lamma, E. (2010b). Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundamenta Informaticae*, 102(3-4):325–361.

Moses, Y. and Tennenholtz, M. (1995). Artificial social systems. *Computers and Artificial Intelligence*, 14(6).

Mouratidis, H., Giorgini, P., Manson, G., Philp, I., et al. (2002). A natural extension of tropos methodology for modelling security. In *the Proceedings of the Agent Oriented Methodologies Workshop (OOPSLA 2002), Seattle-USA*.

Mueller, E. T. (2004). Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, 14(5):703–730.

Mueller, E. T. and Sutcliffe, G. (2005). Discrete event calculus deduction using first-order automated theorem proving. In *Fifth Workshop on the Implementation of Logics*, page 43.

Müller, J. P. and Fischer, K. (2014). Application impact of multi-agent systems and technologies: A survey. In Shehory, O. and Sturm, A., editors, *Agent-Oriented Software Engineering*, pages 27–53. Springer Berlin Heidelberg.

Nakagawa, H., Ohsuga, A., and Honiden, S. (2008). Constructing self-adaptive systems using a kaos model. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 132–137.

Object Management Group (2006). Business process modeling notation specification.

Odell, J. (2002). Objects and agents compared. *Journal of object technology*, 1(1):41–53.

Odell, J., Parunak, H. V. D., and Bauer, B. (2000). Extending uml for agents.

Odell, J. J., Van Dyke Parunak, H., Fleischer, M., and Brueckner, S. (2003). Modeling agents and their environment. In *Proceedings of the 3rd International Conference on Agent-oriented Software Engineering III*, AOSE'02, pages 16–31, Berlin, Heidelberg. Springer-Verlag.

O'malley, S. A. and DeLoach, S. A. (2002). Determining when to use an agent-oriented software engineering paradigm. In Wooldridge, M., Weiss, G., and Ciancarini, P., editors, *Agent-Oriented Software Engineering II*, volume 2222 of *Lecture Notes in Computer Science*, pages 188–205. Springer Berlin Heidelberg.

Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456.

Oreizy, P., Heimbigner, D., Johnson, G., Gorlick, M. M., Taylor, R. N., Wolf, A. L., Medvidovic, N., Rosenblum, D. S., and Quilici, A. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent systems*, 14(3):54–62.

Padget, J., Elakehal, E., Satoh, K., and Ishikawa, F. (2014). On requirements representation and reasoning using answer set programming. In *Artificial Intelligence for Requirements Engineering (AIRE), 2014 IEEE 1st International Workshop on*, pages 35–42.

Padgham, L. and Winikoff, M. (2002). Prometheus: a methodology for developing intelligent agents. In *International Conference on Autonomous Agents & Multiagent Systems*, pages 37–38. ACM.

Padgham, L., Winikoff, M., and Poutakidis, D. (2005). Adding debugging support to the prometheus methodology. *Engineering Applications of Artificial Intelligence*, 18(2):173 – 190. Agent-oriented Software Development.

Papasimeon, M. and Heinze, C. (2001). Extending the uml for designing jack agents. In *Software Engineering Conference, 2001. Proceedings. 2001 Australian*, pages 89–97. IEEE.

Pavón, J. and Gómez-Sanz, J. (2003). Agent oriented software engineering with ingenias. In *Proceedings of the 3rd Central and Eastern European conference on Multi-agent systems*, CEEMAS'03, pages 394–403, Berlin, Heidelberg. Springer-Verlag.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45 – 77.

Pesic, M., Schonenberg, M., Sidorova, N., and Aalst, W. (2007). Constraint-based workflow models: Change made easy. In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94. Springer Berlin Heidelberg.

Pesic, M. and van der Aalst, W. M. (2006a). A declarative approach for flexible business processes management. In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer Berlin Heidelberg.

Pesic, M. and van der Aalst, W. M. (2006b). A declarative approach for flexible business processes management. In *Business Process Management Workshops*, pages 169–180. Springer.

Poutakidis, D., Padgham, L., and Winikoff, M. (2003). An exploration of bugs and debugging in multi-agent systems. In *Proceedings of the 14th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, pages 628–632. ACM Press.

Rao, A. S., Georgeff, M. P., et al. (1995). Bdi agents: From theory to practice. In *International Conference on Multiagent Systems*, volume 95, pages 312–319.

Ricci, A., Denti, E., and Omicini, A. (2001). Agent coordination infrastructures for virtual enterprises and workflow management. In Klusch, M. and Zambonelli, F., editors, *Cooperative Information Agents V*, volume 2182 of *Lecture Notes in Computer Science*, pages 235–246. Springer Berlin Heidelberg.

Robertson, D. (2004). A lightweight method for coordination of agent oriented web services. In *Proceedings of AAAI Spring Symposium on Semantic Web Services*, volume 2.

Rolland, C., Souveyet, C., and Achour, C. B. (1998). Guiding goal modeling using scenarios. *Software Engineering, IEEE Transactions on*, 24(12):1055–1071.

Roy, P. V. (2007a). Self management and the future of software design. *Electronic Notes in Theoretical Computer Science*, 182(0):201 – 217. Proceedings of the Third International Workshop on Formal Aspects of Component Software (FACS 2006).

Roy, P. V. (2007b). Self management and the future of software design. *Electronic Notes in Theoretical Computer Science*, 182:201–217.

Ruiter, D. W. (1997). A basic classification of legal institutions. *Ratio Juris*, 10(4):357–371.

Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (1995). *Artificial Intelligence: A Modern Approach*, volume 2. Prentice Hall.

Sabas, A., Delisle, S., and Badri, M. (2002). A comparative analysis of multiagent system development methodologies: Towards a unified approach. In *In Third International Symposium From Agent Theory to Agent Implementation (AT2AI-3*, pages 599–604.

Sadiq, S. W., Orlowska, M. E., and Sadiq, W. (2005). Specification and validation of process constraints for flexible workflows. *Information Systems*, 30(5):349–378.

Salehie, M. and Tahvildari, L. (2009a). Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14.

Salehie, M. and Tahvildari, L. (2009b). Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42.

Schmeck, H., Müller-Schloer, C., Çakar, E., Mnif, M., and Richter, U. (2010). Adaptivity and self-organization in organic computing systems. *ACM Trans. Auton. Adapt. Syst.*, 5(3):10:1–10:32.

Searle, J. (2010). *The Construction of Social Reality*. Free Press.

Searle, J. R. (1969). *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press.

Shanahan, M. (1999). The event calculus explained. In *Artificial Intelligence Today*, pages 409–430. Springer.

Shehory, O. and Sturm, A. (2001). Evaluation of modeling techniques for agent-based systems. In *Proceedings of the fifth international conference on Autonomous agents*, volume 2001, pages 624–631. ACM.

Shimanoff, S. (1980). *Communication Rules: Theory and Research*. SAGE Library of Social Research. SAGE Publications.

Shoham, Y. (1993). Agent-oriented programming. *Artificial intelligence*, 60(1):51–92.

Silva Souza, V. E. (2012). *Requirements-based Software System Adaptation*. PhD thesis, University of Trento.

Singh, M. P. (1991). Social and psychological commitments in multiagent systems. In *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels*, pages 104–106. AAAI, Inc.

Singh, M. P. (2000). A social semantics for agent communication languages. In *Issues in Agent Communication*, pages 31–45. Springer.

Singh, M. P. and Chopra, A. K. (2010). Correctness properties for multiagent systems. In *Declarative Agent Languages and Technologies VII*, pages 192–207. Springer.

Sinur, J., Odell, J. J., and Fingar, P. (2013). *Business process management: the Next Wave*. Meghan-Kiffer Press.

Smith, H. and Fingar, P. (2003). *Business process management: the third wave*, volume 1. Meghan-Kiffer Press Tampa.

Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, 100(12):1104–1113.

Sokolova, M. V. and Fernandez-Caballero, A. (2010). Supporting multi-agent systems life cycle by integrating protege and prometheus. *International Journal of Intelligent Information and Database Systems*, 4(3):227–244.

Stegers, R., Ten Teije, A., and Van Harmelen, F. (2006). From natural language to formal proof goal. In *Managing Knowledge in a World of Networks*, pages 51–58. Springer.

Sterritt, R., Hinchey, M., and Vassev, E. (2010). Self-managing software. *Encyclopedia of Software Engineering*, pages 1072–1081.

Stonebraker, M., Aoki, P. M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., and Yu, A. (1996). Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):048–063.

Stratulat, T., Ferber, J., and Tranier, J. (2009). Masq: towards an integral approach to interaction. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 813–820. International Foundation for Autonomous Agents and Multiagent Systems.

Sturm, A., Dori, D., and Shehory, O. (2003). Single-model method for specifying multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 121–128. ACM.

Sudeikat, J., Braubach, L., Pokahr, A., and Lamersdorf, W. (2004). Evaluation of agent - oriented software methodologies - examination of the gap between modeling and platform. In Giorgini, P., Müller, J. P., and Odell, J., editors, *Agent-Oriented Software Engineering V, Fifth International Workshop AOSE 2004*, pages 126–141. Springer Verlag.

Susi, A., Perini, A., Mylopoulos, J., and Giorgini, P. (2005). The tropos metamodel and its use. *Informatica (Slovenia)*, 29(4):401–408.

Tampitsikas, C., Bromuri, S., Fornara, N., and Schumacher, M. I. (2012). Interdependent artificial institutions in agent environments. *Applied Artificial Intelligence*, 26(4):398–427.

Tamura, G., Villegas, N. M., Müller, H. A., Sousa, J. P., Becker, B., Karsai, G., Mankovskii, S., Pezzè, M., Schäfer, W., Tahvildari, L., et al. (2013). Towards practical runtime verification

and validation of self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, pages 108–132. Springer.

Tanenbaum, A. S. and van Steen, M. (2002). *Distributed systems*, volume 2. Prentice Hall Upper Saddle River.

Therborn, G. (2002). Back to norms! on the scope and dynamics of norms and normative action. *Current Sociology*, 50(6):863–880.

Tran, Q.-N. N., Low, G., and Williams, M.-A. (2003). A feature analysis framework for evaluating multi-agent system development methodologies. In *Foundations of Intelligent Systems*, pages 613–617. Springer.

Tran, Q.-N. N., Low, G., and Williams, M.-A. (2005). A preliminary comparative feature analysis of multi-agent systems development methodologies. In *Agent-Oriented Information Systems II*, pages 157–168. Springer.

Tuomela, R. and Bonnevier-Tuomela, M. (1995). Norms and agreement. *European Journal of Law, Philosophy and Computer Science*, 5:41–46.

van der Aalst, W. and Pesic, M. (2006). Decserflow: Towards a truly declarative service flow language. In et al, F. L., editor, *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany.

van der Aalst, W. M., De Beer, H., and van Dongen, B. F. (2005). *Process mining and verification of properties: An approach based on temporal logic*. Springer.

van der Aalst, W. M., van Dongen, B. F., Günther, C. W., Mans, R., De Medeiros, A. A., Rozinat, A., Rubin, V., Song, M., Verbeek, H., and Weijters, A. (2007). Prom 4.0: comprehensive support for real process analysis. In *Petri Nets and Other Models of Concurrency–ICATPN 2007*, pages 484–494. Springer.

Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 249–262. IEEE.

van Riemsdijk, M. B., Hindriks, K. V., and Jonker, C. (2009). Programming organization-aware agents. In Aldewereld, H., Dignum, V., and Picard, G., editors, *ESAW*, volume 5881 of *Lecture Notes in Computer Science*, pages 98–112. Springer.

Vázquez-Salceda, J. (2003). The role of norms and electronic institutions in multi-agent systems applied to complex domains. the harmonia framework. *Ai Communications*, 16(3):209–212.

Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., and Becker, B. (2009). Model-driven architectural monitoring and adaptation for autonomic systems. In *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, pages 67–68, New York, NY, USA. ACM.

von Wright, G. H. (1963). *Norm and Action: A Logical Enquiry*. Routledge and Kegan Paul.

Weske, M. (2012). *Business process management: concepts, languages, architectures*. Springer.

Westergaard, M. and Maggi, F. (2012). Looking into the future. In Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., and Cruz, I., editors, *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7565 of *Lecture Notes in Computer Science*, pages 250–267. Springer Berlin Heidelberg.

Weyns, D. and Georgeff, M. (2010). Self-adaptation using multiagent systems. *Software, IEEE*, 27(1):86–91.

Winikoff, M. and Padgham, L. (2004). *Developing Intelligent Agent Systems: A Practical Guide*. Halsted Press, New York, NY, USA.

Wood, B., Pethia, R., Gold, L. R., and Firth, R. (1988). A guide to the assessment of software development methods. Technical report, DTIC Document.

Wood, M. and Deloach, S. A. (2000). An overview of the multiagent systems engineering methodology. In *The First International Workshop on Agent-Oriented software Engineering (AOSE-2000*, pages 207–222.

Wooldridge, M. (1997). Agent-based software engineering. *IEE Proceedings-software*, 144(1):26–37.

Wooldridge, M. (2008). *An introduction to multiagent systems*. Wiley. com.

Wooldridge, M. and Jennings, N. R. (1995a). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152.

Wooldridge, M. and Jennings, N. R. (1995b). Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02):115–152.

Wooldridge, M. and Jennings, N. R. (1998). Pitfalls of agent-oriented development. In *Proceedings of the second international conference on Autonomous agents*, pages 385–391. ACM.

Wooldridge, M., Jennings, N. R., and Kinny, D. (2000a). The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3:285–312.

Wooldridge, M., Jennings, N. R., and Kinny, D. (2000b). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312.

Wooldridge, M. J. (1992). *On the Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, UMIST, Department of Computation, Manchester, UK.

Yeom, K. and Park, J.-H. (2012). Morphological approach for autonomous and adaptive systems based on self-reconfigurable modular agents. *Future Generation Computer Systems*, 28(3):533 – 543.

Yolum, P. and Singh, M. P. (2001). Designing and executing protocols using the event calculus. In *Proceedings of the fifth international conference on Autonomous agents*, pages 27–28. ACM.

Yu, E. (2009). Social modeling and i*. In Borgida, A., Chaudhri, V., Giorgini, P., and Yu, E., editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 99–121. Springer Berlin Heidelberg.

Yu, E. (2011). Modelling strategic relationships for process reengineering. *Social Modeling for Requirements Engineering*, 11:2011.

Yu, E. S. K. and Mylopoulos, J. (1994). Using goals, rules, and methods to support reasoning in business process reengineering. In *in Business Process Reengineering, International Journal of Intelligent Systems in Accounting, Finance and Management*, pages 1–13.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2001). Organisational rules as an abstraction for the analysis and design of multi-agent systems.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003a). Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003b). Developing multiagent systems: The gaia methodology.